

Deep Visual Instruments:

Realtime Continuous, Meaningful Human Control
over Deep Neural Networks for Creative Expression

Mehmet Selim Akten



A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Goldsmiths, University of London
Department of Computing

April 2021

I, Mehmet Selim Akten, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed:

Date: 22 April, 2021

Abstract

In this thesis, we investigate Deep Learning models as an artistic medium for new modes of performative, creative expression. We call these *Deep Visual Instruments*: realtime interactive generative systems that exploit and leverage the capabilities of state-of-the-art Deep Neural Networks (DNN), while allowing *Meaningful Human Control*, in a *Realtime Continuous manner*.

We characterise *Meaningful Human Control* in terms of intent, predictability, and accountability; and *Realtime Continuous Control* with regards to its capacity for performative interaction with immediate feedback, enhancing goal-less exploration. The capabilities of DNNs that we are looking to exploit and leverage in this manner, are their ability to learn hierarchical representations modelling highly complex, real-world data such as images. Thinking of DNNs as tools that extract useful information from massive amounts of Big Data, we investigate ways in which we can navigate and explore *what* useful information a DNN has learnt, and *how* we can meaningfully use such a model in the production of artistic and creative works, in a performative, expressive manner.

We present five studies that approach this from different but complementary angles. These include: a collaborative, generative sketching application using MCTS and discriminative CNNs; a system to gesturally conduct the realtime generation of text in different styles using an ensemble of LSTM RNNs; a performative tool that allows for the manipulation of hyperparameters in realtime while a Convolutional VAE trains on a live camera feed; a live video feed processing software that allows for digital puppetry and augmented drawing; and a method that allows for long-form story telling within a generative model's latent space with meaningful control over the narrative.

We frame our research with the realtime, performative expression provided by musical instruments as a metaphor, in which we think of these systems as not *used* by a *user*, but *played* by a *performer*.



Acknowledgements

It would be an understatement to call these last few years that I have been working on this thesis, an incredible journey. As I reflect back on these years, especially in the middle of a global pandemic and crisis, I am truly moved to recognize how fortunate I have been with regards to the incredible support that I have received from a truly remarkable network of people.

First and foremost, I cannot thank my supervisors Professor Mick Grierson and Dr. Rebecca Fiebrink enough for their unwavering guidance over these years. I am indebted to your faith and patience; and your invaluable mentorship in showing me how to think and organise my thoughts. Thank you both for making this PhD an incredibly enjoyable experience. I am especially indebted to my primary supervisor Mick. Thank you for giving me the space, the time, the freedom, and the opportunities to explore, to discover, to fail, to get lost; and then knowing exactly when and how to signal me back. Professor Frederic Fol Leymarie, though you were not officially my supervisor, your ongoing guidance and feedback has been invaluable, always so incredibly concise and clear. I owe an additional thanks to Professor Atau Tanaka, and Professor William Latham for your ongoing support, and in particular to Atau, for talking me into applying for this PhD position. I am sincerely grateful to the UK's EPSRC whose funding supported my research, and to Professor Peter Cowling, Professor Simon Lucas, Dr Jeremy Gow, Jo Maltby, Lucy Jeczalik, Tuula Juvonen and everybody else involved in IGGI, for making this experience not only possible, but as smooth as it could be.

I would also like to extend my gratitude to my PhD and lab companions: Daniel Berio, Tom Cole, Tara Collingwoode-Williams, Janet Gibbs, Christian Guckelsberger, Rob Homewood, Zoë O'Shea. You always made it a joy coming into the lab. In particular I would like to thank Christian, for the stimulating conversations around creativity, curiosity, intrinsic motivation, information theory and beyond; Rob, for the limitless energy and support on so many levels, on our adventures in Eindhoven, in Prague, in Transylvania; and Daniel, for our collaborations, the late nights trying to hit submission deadlines, and for opening my mind to the Art of Graffiti, leading me into the rabbit holes of motor control and cognition, and embodied simulation.

Much of the work in this thesis was conducted from my studio in Somerset House Studios. Jonathan Reekie, Marie McPartlin, Emma Hannon, Stella Sideli, Leonara Manyangadze, you are awesome. You have built an incredible space, and an incredible community. I am honoured to have been part of it, and I thank you for your support over these years. My Somerset House neighbours: Alan Warburton, Anab Jain & Jon Ardern and the rest of the Superflux crew, Anna Meredith, Carmen Aguilar y Wedge, Estela Oliva, Libby Heaney, Marija Bozinovska Jones, Matthew Plummer-Fernandez, Nick Ryan, what a stellar group of people you are, I miss you all so much! I am eternally grateful to my SHS collaborators Jennifer Walshe and Jenna

Sutela. You pushed my mind to places it would never have thought to go, and I am incredibly proud of the work we have done. I would like to extend an additional thanks to the Chase Foundation, for their support in my collaboration with Jennifer Walshe; and the Serpentine Gallery and Google Arts & Culture for their support in my collaborations with Jenna Sutela. I owe an additional thanks to Google’s Artists & Machine Intelligence program, for offering me the opportunity to be one of their first Artists-in-residence, and then allowing me to use that opportunity to investigate the stereotypes and bias’s embedded in Google’s language models. In that respect I would like to extend an extra thanks to Mike Tyka and Kenric Mcdowell, for not only making this opportunity happen, but for the endless fascinating conversations, in so many different corners of the world, from Tokyo to London to Seattle.

This thesis would not have been possible without the long-lasting support from my long-term collaborators, my friends, my Hackney Wick posse: Alexander Whitley, Barney Steel, Clemmie Mason, Davide Quayola, Ersin Han Ersin, Giorgia Polizzi, Nell Whitley, Robin McNicholas, Ruairi Glynn, Sandra Ciampone, and Tim Exile. We have been on this journey for over a decade. The endless discussions and investigations into visual instruments, realtime modes of interaction and creativity, perception, embodied cognition; over Sunday Roasts, Crate pizza and beer by the canal — these have all played a central role in shaping me and my thoughts on this subject, and in this thesis. I miss you all. And I’d like to extend a heartfelt thank you to Jane Laurie, for your immeasurable amounts of patience and support during these years. Ruby, Pearl, Bruce, Jeff Bridges, you brought so me much joy. I miss you.

A special thanks goes to the wonderful folks at Stochastic Labs in Berkeley, my most recent family who supported me leading up to my final push: Vero Bollow, Primavera De Filippi, Joel Simon, Alex Reben, Steve Thompson, Aza Raskin, Thank you for giving me a place to focus and work, and a beautiful home. You have been a wonderful family and I miss you all.

I am deeply indebted to the curators, festival organisers, galleries, producers and writers who supported my work by getting it out there and situating it in wider context. I would like to thank so many inspiring artists and researchers working in this field, many of whom are cited in this thesis, and I’d like to especially thank you for sharing your code :). In this capacity I’d like to extend a personal thank you to the following people, my colleagues, friends, not only for your inspiring work and visions, but for your support, and the thought-provoking and insightful conversations which has helped guide me in this time: Alexander Mordintsev, Allison Parish, Angelique Spaninks, Anna Ridler, Behnaz Farahi, Caspar Sonnen, Catherine Griffiths, Christiane Paul, Christine Schöpf, Christl Baur, Damien Henry, Daniel Shiffman, David Ha, David OReilly, David Rokeby, Daria Parkhomenko, Drew Hemment, Doug Eck, Elliot Woods, Emiko Ogawa, Eva Jaeger, Gaby Jenkins, Gene Kogan, Golan Levin, Gerfried Stocker, Hannah Davis, Honor Harger, Irimi Papadimitriou, Jérôme Neutres, Jesse Engel, Joanie Lemercier, Joel Gethin Lewis, Jonathan Harris, José Luis de Vicente, Juliette Bibasse, Juliette Larthe, Karina Smigla-Bobinski, Kyle Kastner, Kyle Mcdonald, Lalin Akalan, Lauren McCarthy, Liam Young, Luba Elliott, Lucy McRae, Manuela Naveau, Martin Honzik, Max Cooper, Mario Klingemann, Marjan Sharifi, Mira Calix, Moco Ziegler, Parag Mital, Rachel Uwa, Refik Anadol, Renee Zachariou, Ross Goodwin, Sander Dieleman, Şerife Wong, Shane & Sophie Walter, Tom Higham, Tom White. And a special thanks to Hanna Radek, for keeping me sane during the final push.

I would like to pay a special tribute to JT Nimoy. You have been such a huge inspiration to me, and to so many people. You are deeply influential on my thinking about the topics in this thesis. You are a visionary and a pioneer. You were too ahead of your time. The world was not ready for you.

Finally, I would like to thank my family. Due to the unexpected turn of events that took place in 2020, we ended up spending more time together than we have done since the 1990s. For this experience, I am eternally grateful. Writing my thesis this summer, amidst a global pandemic no less, could have been an incredibly stressful period. And in many ways it has been. But it has also become one of the most treasured periods of my life. I am beyond grateful for making me feel at home, once again after all these years. Thank you to my mother Nur, my father Tuncer, my sister Zeynep, and of course little Nora.



Contents

Abstract	iii
Acknowledgements	v
List of Figures	xiii
List of algorithms	xxiii
1 Introduction & Motivations	1
1.1 Background	1
1.2 Why Deep Learning?	4
1.3 Meaningful Human Control	5
1.3.1 Pressing a button	6
1.3.2 ‘Random’ faders	7
1.3.3 Necessary and sufficient conditions	7
1.4 Visual instruments: Realtime Continuous Control	8
1.4.1 Realtime Continuous Control	8
1.4.2 Visual instruments	8
1.4.3 Realtime performative interaction	9
1.4.4 Goal-less exploration	9
1.4.5 Flow	10
1.5 Creative DL × Meaningful & Realtime Continuous Control	11
1.5.1 The State	11
1.5.2 The Problem	11
1.5.3 The Reason	11
1.6 Conclusion: why is this important	12
1.7 Research	13
1.7.1 Research summary	13
1.7.2 Research methods and evaluation	14
1.8 Summary of contributions and impact	15
1.8.1 Press	17
1.8.2 Invited presentations and panels	17
1.8.3 Public showings	18
1.8.4 Opensource software	19

1.9	Thesis outline	20
2	Background	23
2.1	Generative models	24
2.1.1	Unconditional generative models	25
2.1.2	Conditional generative models	26
2.1.3	Latent manipulations	26
2.1.4	‘Generative’ terminology in different domains	29
2.2	Very brief histories	30
2.2.1	Generative art	31
2.2.2	AI and ML in art, pre-Deepdream	31
2.2.3	Interactive media art	32
2.2.4	Visual instruments	33
2.2.5	Convergence	33
2.2.6	Creative DL \leftarrow AI Art \cup Creative AI	33
2.2.7	Creative Deep Learning — from a cultural perspective	34
2.2.8	Computational Creativity	46
2.2.9	Machine Learning for Artistic, Expressive Human Computer Interaction (AE-HCI)	47
2.3	Introduction to Deep Learning	51
2.3.1	Overview	51
2.3.2	Machine Learning (ML)	52
2.3.3	Artificial Neural Networks (ANN)	52
2.3.4	Feed-forward (FNN) vs Recurrent Neural Networks (RNN)	53
2.3.5	Layers	53
2.3.6	Multi-Layer Perceptrons (MLP)	53
2.3.7	Universal function approximators, expressive power	54
2.3.8	Learning	55
2.3.9	Loss functions	55
2.3.10	Gradient descent and backpropagation	56
2.3.11	Optimisation algorithms	57
2.3.12	Deep Neural Networks (DNN)	57
2.3.13	Hyperparameter search	58
2.3.14	Classes of learning	59
2.3.15	Convolutional Neural Networks (CNN)	60
2.3.16	Auto-Encoders (AE)	63
2.3.17	Variational Auto-Encoders (VAE)	64
2.3.18	Deep Convolutional Generative Adversarial Networks (DCGAN)	67
2.3.19	Recurrent Neural Networks (RNN)	69
2.3.20	Monte Carlo Tree Search	73
2.4	Conclusion	75

3	Realtime sequence generation with continuous control	77
3.1	Introduction	77
3.2	Collaborative generative sketching with MCTS and CNNs	79
3.2.1	Introduction	79
3.2.2	Background	79
3.2.3	Overview	80
3.2.4	System description	81
3.2.5	Results and discussion	84
3.3	Realtime interactive text generation with RNN ensembles	88
3.3.1	Introduction	88
3.3.2	Background	88
3.3.3	Overview	90
3.3.4	System description	94
3.3.5	Results and discussion	95
3.4	Conclusion	95
4	Hello World: Realtime interactive training as an informative and performative tool	97
4.1	Introduction	97
4.2	Motivations	98
4.3	Background	100
4.4	System description	104
4.4.1	Hyperparameters	107
4.4.2	Batch size, exponentially decaying memory and augmentation	108
4.5	Experiments and results	109
4.5.1	Optimiser and associated hyperparameters	109
4.5.2	Reconstruction loss	114
4.5.3	Latent loss and variational reparametrisation	115
4.5.4	Video feed manipulations	117
4.6	Conclusions	121
5	Learning to see: Digital puppetry through realtime video transformation	123
5.1	Introduction	123
5.2	Overview	125
5.3	System description	133
5.3.1	Datasets	133
5.3.2	Training	134
5.3.3	Inference	139
5.4	Experiments and results	140
5.4.1	Augmented drawing	140
5.4.2	Digital puppetry	144
5.4.3	Live parameter manipulation	151
5.5	Conclusion	168

6	Deep Meditations: Latent storytelling	173
6.1	Introduction	173
6.2	Background and motivations	175
6.3	Goal and requirements	176
6.3.1	Main tasks	176
6.3.2	Additional issues	177
6.4	System description	178
6.4.1	Concepts and definitions	179
6.4.2	Exploration of the model	183
6.4.3	The narrative edit	184
6.4.4	The narrative conform	185
6.4.5	Trajectory planner	186
6.4.6	The final render	186
6.4.7	Model architecture and data	186
6.4.8	Trajectory planner details	187
6.5	Conclusion	193
7	Conclusion	195
7.1	Summary of research background and objectives	195
7.2	Research methodology	197
7.3	Summary of contributions and outcomes	199
7.4	Future directions	204
7.5	Final thoughts — Human-Machine Collaboration	208
	References	211

List of Figures

1.1	The hand stencils at the <i>Cuevas de las Manos</i> in Santa Cruz, Argentina, are thought to be over 10,000 years old. Image from Wikimedia Commons by User:Marianocecowski, licenced under CC BY-SA 3.0.	1
2.1	Still image from video “ <i>Deepdream is blowing my mind</i> ” (2015) by Memo Akten.	36
2.2	“ <i>All watched over by machines of loving grace: Deepdream edition</i> ” (2015) at the “ <i>DeepDream: The art of Neural Networks</i> ” exhibition at the Gray Area Foundation, February 2016.	37
2.3	A selection of press covering the “DeepDream: The art of Neural Networks” exhibition at the Gray Area Foundation, February 2016.	39
2.4	a.) A three layer, feed-forward, Multi-Layer Perceptron with a single hidden layer. In this diagram, all of the neurons are displayed, and we can see that the network consists of 4 inputs and 3 outputs, and a 5 neuron hidden layer. b.) The same MLP, displayed with layers collapsed into multi-variate <i>units</i> allowing simpler visualisation, mathematical modelling and computational implementation using vectors. The dimensions of each layer is indicated with <i>dim</i> , \mathbf{W}_{xh} and \mathbf{W}_{hy} denote the weights of the connections from the input layer to the hidden layer, and hidden layer to the output layer respectively, \mathbf{b}_h and \mathbf{b}_y denote the bias vectors (which are typically included as parts of the weights or parameters) of the hidden layer and output layer respectively, and φ denotes the activation functions. c.) A Recurrent Neural Network, with a cyclic connection on the hidden layer.	54
2.5	A Convolutional layer consisting of many filters, applied to an input. Image from Wikimedia Commons, licensed under CC BY-SA 4.0	61
2.6	Image from Wikimedia Commons, licensed CC under BY-SA 4.0	62
2.7	A simple Auto-Encoder. In this example, the input and output layers consist of 10 neurons, while the bottleneck layer consists of 3 neurons. To avoid clutter in the diagram, we have omitted drawing all of the connections. In a MLP Auto-Encoder, all of the layers are fully connected. In a Convolutional Auto-Encoder, the layers are arranged into grids as we describe in subsection 2.3.15: <i>Convolutional Neural Networks (CNN)</i>	64

2.8	Variational Auto-Encoder. The output of the encoder network is the mean μ and standard deviation σ of a Normal distribution, from which a latent vector z is sampled $z \sim \mathcal{N}(\mu, \sigma)$. This is then fed into the decoder network for decoding.	65
2.9	Generative Adversarial Network	68
2.10	MLP and RNN with single hidden layer.	70
2.11	Overview of the steps of MCTS	75
3.1	Overview of how MCTS and the CNN is integrated	82
3.2	A screenshot from our software, running the MLR model trained on MNIST. The top left panel shows the current state of the canvas for the current timestep, the top middle panel shows the scaled down drawing used for evaluation, and the top right panel shows a visualization of all of the current MCTS rollouts for the current timestep. At the bottom of the screenshot, the class probability distribution shows that the classifier returns 100% confidence that the current state of the drawing is the digit ‘3’.	84
3.3	A screenshot of our software, running the Inception-v3 model trained on ImageNet with desired target class set to ‘meerkat’. At the bottom of the screenshot, the class probability distribution shows that the classifier is 100% confident that the drawing is of a ‘meerkat’.	86
3.4	The same configuration as Fig. 3.3 except the desired target class is ‘white wolf’, and the class probability distribution shows that the classifier is 98% confident that the drawing is of a ‘white wolf’.	86
3.5	Screenshot from our dual-screen interactive Char-RNN ensemble software. On the left is the <i>interaction screen</i> which we typically display on a touchscreen monitor. On the right is the <i>output screen</i> where the text is displayed as it is generated character-by-character.	90
3.6	The two screens from Fig. 3.5 stacked vertically for ease of reading in print. . . .	91
3.7	Software architecture for our interactive Char-RNN. The <i>Server</i> is a python-based console application with no Graphical User Interface, that manages all of the models. At every timestep, the Server receives a <i>seed</i> text from the <i>Visualiser</i> , runs it through all of the models, and returns all of the probability distributions output from each model to the Visualiser. The Visualiser is an OpenGL application which provides the visual interface and interaction. The screenshots in Fig. 3.5 and Fig. 3.6 are from the Visualiser. The Visualiser continually tracks the performer’s actions, either via mouse input, tracking the performer’s hands using a LeapMotion device, or via faders on an external midi controller. At every timestep, the Visualiser uses these inputs to calculate and update style mixture weights, which are used to calculate the <i>joint probability distribution</i> for the next character. The Visualiser samples the next character from this distribution, updates the screen with this new character, and also sends it back to the Server via OSC, so that the Server can update the internal state of each of the models. . . .	94

4.1	Hello World at <i>The Moscow Museum of Modern Art</i> , 2018 (Photo © Yuri Palmin)	99
4.2	Variational Auto-Encoder. The output of the encoder network is the mean μ and standard deviation σ of a Normal distribution, from which a latent vector z is sampled $z \sim \mathcal{N}(\mu, \sigma)$. This is then fed into the decoder network for decoding.	101
4.3	An adaptation of <i>Hello World</i> at “ <i>Artists and Robots</i> ”, Astana Contemporary Art Centre, 2017	103
4.4	A screenshot from our <i>Hello World</i> software.	106
4.5	A selection of frames covering roughly a two minute experimentation session where we play with the optimisation hyperparameters learning rate, momentum, and gradient clipping threshold. Time flows from left to right, and each frame shows the live reconstruction, i.e. the <i>Seeing</i> panel.	112
4.6	Six screenshots showing a qualitative comparison of the four different loss functions. Four models train in parallel, with identical architecture and hyperparameters and the only difference being the loss functions. MS-SSIM converges the quickest, producing the sharpest reconstructions, while L2 and CE loss produce the most blurry.	115
4.7	Three screenshots taken during the first 30 seconds of a new experimentation session. The model is initialised at the beginning, and by the 3rd row, has converged to the extent that the reconstructed image <i>Seeing</i> is sharp and resembles the live video feed <i>Stimulus</i> . This indicates that the model has become familiar with, and has learnt the necessary features required to represent and reconstruct this <i>Stimulus</i> image.	117
4.8	In the top row, we hold our left arm vertical and hold it still for a few seconds while we wait for the reconstructed image <i>Seeing</i> to become sharp. We then rotate our arm to be horizontal. As can be seen in the 2nd row, the entire reconstructed image <i>Seeing</i> is sharp, except for our arm, which is missing. The model has not seen our arm in a horizontal position yet, and thus lacks the representations required to visualise it in that manner. Across the last two rows, the arm flickers and fades in as the model learns the necessary representations for the arm in this position. Note that in the 3rd column <i>Reminiscing</i> , the arm flickers between horizontal and vertical. This is due to the fact that the <i>Reminiscing</i> image is a random sample from the local neighbourhood of the <i>Stimulus</i> . And in the well structured latent space of a generative model, this local neighbourhood will contain images that are aesthetically, structurally, and/or semantically similar. For this reason, the <i>Reminiscing</i> image will be random samples that <i>resemble</i> the <i>Stimulus</i> , without necessarily being straight reconstructions.	118

4.9	In the top row, we hold a red phone in our right hand for a few seconds while we wait for the reconstructed image <i>Seeing</i> to become sharp. In the second row, we put the phone down, and quickly lift our hand to the same position. In <i>Seeing</i> , trying to reconstruct this image of our hand without a phone, the model samples the <i>closest image</i> it can to this <i>Stimulus</i> . And that is limited by what it has seen before, and the representations that it has learnt. In this case, the red phone appears in our hand. Over the course of the subsequent rows, the phone flashes on and off and gradually fades out as the model learns the necessary representations for this <i>Stimulus</i> . In the <i>Reminiscing</i> panel however, we can see that the local neighbourhood of this point in latent space is still dominated by the red phone. It will take another minute or two for the distribution to be evened out across this larger area in latent space.	119
4.10	We hold the red phone in our left hand. However the model has not seen this. In fact it has only seen our arm in this position with our hand open and flat, as it was in Fig. 4.8 . So in <i>Seeing</i> , this is how the model reconstructs this <i>Stimulus</i> image. Over the next few frames, the red phone gradually fades in, flickering in and out. After the 5th frame, we start slowly rotating our arm back and forth between horizontal and vertical, and we observe the results in both <i>Seeing</i> , and <i>Reminiscing</i> as the phone flickers in and out, and the arm even snaps to horizontal or vertical positions.	120
5.1	Frames from <i>Gloomy Sunday</i> (video, 2017), demonstrating subsection 5.4.2: <i>Digital puppetry</i> . Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.	127
5.2	Frames from <i>Gloomy Sunday</i> (video, 2017), demonstrating subsection 5.4.2: <i>Digital puppetry</i> . Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.	128
5.3	Frames from a video demonstrating subsection 5.4.1: <i>Augmented drawing</i> . Each pair of images shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.	129

5.4	Frames from <i>We are all made of star dust</i> (video, 2017), demonstrating subsection 5.4.3: <i>Live parameter manipulation</i> . In the top frame, we adjust the parameters such that the desired output brightness is low. The system automatically uses dark features to construct the output image. In this case, most of the output image is comprised of distant galaxies, while hints of larger and brighter distant galaxies define the outline of the face in the video feed. In the lower two frames, we increase the desired brightness, and the system automatically replaces the distant galaxies with brighter features, such as nebulae. We demonstrate this in much more detail in later sections.	130
5.5	Frames from a video demonstrating <i>multiple simultaneous models</i> . This was presented as an <i>interactive installation</i> at <i>International Documentary Film Festival Amsterdam</i> (IDFA) 2017. Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the subsequent panels show images generated in realtime by our software using multiple models trained on different datasets.	131
5.6	section 5.4.2: <i>Interactive installation</i> as exhibited at The Barbican’s 2019 <i>AI: More than Human</i> exhibition in London, UK and currently on tour around the world. We discuss this in more detail in section 5.4.2: <i>Interactive installation</i> . . .	132
5.7	A high level schematic of the system including the training and inference stages. During pix2pix training, the images which are to be used as <i>inputs</i> in the (<i>input, target</i>) pairs, are generated on the fly from the <i>target</i> images via a custom image processing pipeline. The parameters of this pipeline are randomised on every iteration, in order to augment the dataset and aid generalisation. During inference, the video feed is also fed through the same image processing pipeline. In addition, a number of additional image processing filters are applied before the images are fed into the model for prediction. This exposes a number of human-understandable parameters for realtime manipulation and meaningful control over the images generated.	134
5.8	Example images from <i>training</i> . Each row shows examples from a separate model. From top to bottom, these are the <i>Waves</i> , <i>Flowers</i> and <i>Deep space</i> models. The left column shows an image selected at random from the training data. This is a random crop, and may or may not have been flipped. This will be used as a <i>target</i> image for this particular training iteration. I.e. it is the <i>ground truth</i> . The centre column shows the <i>input</i> image generated on the fly via realtime processing of the <i>target</i> image, with randomised parameters (i.e. random brightness and contrast). The right column shows a prediction from the generative model, given only <i>input</i> . In other words, the model is learning to predict the first column, given only the second column, and is producing the third column.	138

5.9	A screenshot from our realtime inference software with multiple models and the GUI enabled. In the top left corner, we can see the live video feed, either from a camera, or a video file. In the top centre, we can see the video feed processed by the image processing pipeline. This is the <i>input</i> image (labelled <i>capture_proc</i> in the screenshot) that is fed to the trained model(s) for inference. In this particular case, the video feed has been <i>downscaled</i> 24x, and then upscaled back to its original size. This acts as a low-pass filter. The next four panels show the outputs from four different models, simultaneously running inference on the same input. On the far right, we can see the GUI, showing the parameters which are adjustable in realtime. Notice how the original video feed in the top left, is radically different from the raw training examples in the left-most column of Fig. 5.8. However, the centre image, which is actually <i>input</i> into the model during inference, very much resembles the types of images in the centre column of Fig. 5.8, which are input into the model during training.	139
5.10	Frames from a video demonstrating <i>augmented drawing</i> (2017).	140
5.11	Frames from a video demonstrating <i>augmented drawing</i> (2018). More frames from this video can be seen in 5.3.	140
5.12	Stills from <i>Delusions</i> (video, 2017) by French artist and roboticist Patrick Tresset. Each pair of images shows a single frame from the video. In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image output from his software in realtime.	142
5.13	Stills from <i>Body Network on Paper I and II</i> (video, 2019) by American artist and human anatomy expert Scott Eaton.	143
5.14	Frames from the video <i>Gloomy Sunday</i> (2017)	145
5.15	Frames from the video <i>We are made of star dust #2</i> (2017). The live video feed for this example comes from a USB microscope.	146
5.16	Objects provided with the <i>Learning to see</i> interactive installation. Users play with and manipulate these objects. The video feed from an overhead camera is fed to our system for processing.	147
5.17	<i>Learning to see</i> interactive installation at The Barbican’s 2019 <i>AI: More than Human</i> exhibition in London, UK.	147
5.18	<i>Learning to see</i> interactive installation at The Barbican’s 2019 <i>AI: More than Human</i> exhibition in London, UK.	148
5.19	<i>Learning to see</i> interactive installation at The Barbican’s 2019 <i>AI: More than Human</i> exhibition in London, UK.	149
5.20	<i>Learning to see</i> interactive installation at The Barbican’s 2019 <i>AI: More than Human</i> exhibition in London, UK.	150
5.21	Frames from <i>Architectural machine translation</i> (video, 2019) by Swedish architect Erik Swahn. In 2019, inspired by the work that we shared in this area, Swahn started experimenting with a similar approach. He trained models on architectural plans and urban maps. Using wooden blocks, he was able to instantly prototype potential architectural diagrams.	150

5.22	The image here in the top left shows the frozen video feed that will be used as an input for all of the subsequent examples in this section. All of the variations in output in the subsequent examples, come only from different configurations of parameters, which we will detail in each example.	152
5.23	A selection of outputs from the <i>Waves</i> model, collected in a single image to aid side-by-side visual comparison. Note that each of these images was generated from the <i>same</i> input video frame. Only the processing pipeline parameters are interactively modified in realtime via the GUI.	153
5.24	Same configuration as Fig. 5.23, with outputs from the <i>Fire</i> model.	154
5.25	Same configuration as Fig. 5.23, with outputs from the <i>Clouds</i> model.	155
5.26	Same configuration as Fig. 5.23, with outputs from the <i>Deep space</i> model.	156
5.27	<i>downscale=24, gamma=2</i>	157
5.28	<i>downscale=24, gamma=5</i>	157
5.29	More examples of modifying <i>gamma</i> , arranged in a single image to aid side-by-side visual comparison. Each of the columns show the outputs for <i>gamma=2, 3, 5</i> and <i>7</i> respectively, with <i>downscale=24</i> for all.	158
5.30	The first column is for reference, and shows the outputs with <i>invert</i> turned off and <i>gamma=2</i> . The following three columns show the outputs with <i>invert</i> turned on and <i>gamma=1, 2</i> and <i>3</i> respectively.	159
5.31	<i>invert=True, downscale=0., post_median=20, gamma=1</i>	160
5.32	<i>invert=True, downscale=10, post_median=30, gamma=1</i> . In the previous example, to demonstrate the effect of the <i>median</i> filter, we disabled the <i>downscale</i> low-pass filter. This results in undesirable high frequency artefacts in the output images. To address this, we reintroduce a small amount of <i>downscale</i>	160
5.33	More examples of modifying the amount of the <i>median</i> filter, arranged in a single image to aid side-by-side visual comparison. The first column is for reference, and shows the outputs with <i>median</i> turned off, <i>gamma=2</i> and <i>downscale=24</i> . The following three columns show the outputs for <i>median=10, 20</i> and <i>30</i> respectively. The last column also has <i>downscale=10</i> to remove the high frequency artefacts, while the second and third columns have <i>downscale</i> turned off.	161
5.34	<i>adaptive_thresh=True, downscale=0</i> . For each pixel, we set a threshold equal to a gaussian-weighted sum of the pixel's neighbours within a block, minus a constant <i>C</i> . If the pixel value is greater than this threshold, it is set to black, otherwise it is set to white. This behaviour can be inverted with the <i>invert</i> parameter. The block size and constant <i>C</i> can be adjusted with the <i>adaptive_thresh_block</i> and <i>adaptive_thresh_c</i> parameters respectively.	162
5.35	<i>adaptive_thresh=True, downscale=16</i> . In the previous example, we disabled the low-pass filter, to demonstrate the effect of the adaptive threshold. The undesirable high frequency artefacts which arise as a result of this can be resolved with a low-pass filter such as <i>downscale</i>	162

5.36	<i>adaptive_thresh=True, downscale=16, post_median=10</i> . We can reintroduce a <i>median</i> at the end of the image processing pipeline, smoothing the edges of the shapes.	163
5.37	<i>adaptive_thresh=True, downscale=16, post_median=20</i> . Increasing the amount of <i>median</i> smooths the shapes of the structures even more.	163
5.38	The first two columns show the outputs for <i>post_median=10</i> and <i>20</i> respectively. The last two columns show the outputs for <i>pre_median=10</i> and <i>20</i> respectively. In all cases, the <i>adaptive threshold</i> and <i>downscale</i> settings are the same as the previous examples.	164
5.39	<i>canny=True, canny_t1=10, canny_t2=20</i>	165
5.40	<i>canny=True, canny_t1=1, canny_t2=10</i> . We can adjust the threshold values for the canny algorithm using <i>canny_t1</i> and <i>canny_t2</i> to increase or decrease the amount of detail captured.	165
5.41	<i>pre_median=10, canny=True, canny_t1=1, canny_t2=10</i> . Applying a <i>median</i> filter before the canny filter, removes some of the noise and smooths the edges detected by the algorithm.	166
5.42	<i>pre_median=10, canny=True, canny_t1=1, canny_t2=10, post_blur=20</i> . As we demonstrated in the previous examples, to remove the high frequency artefacts in the output images, we can apply a low-pass filter to the input image. In the previous examples, we used <i>downscale</i> as a low-pass filter. Another option that we have available, is to use a <i>blur</i> . The difference between <i>post_blur</i> and <i>downscale</i> is very subtle, but for the sake of variety, we will use <i>blur</i> in these next few examples. Similar to <i>median</i> , we can apply the blur at the start of the filter pipeline, or at the end. In this particular case, since we would like to filter the results of the canny edge detection, we use <i>post_blur</i>	166
5.43	More examples of modifying the <i>canny</i> thresholds and the amounts of <i>pre_median</i> and <i>post_blur</i> filters. The (<i>pre_median, canny_t1, canny_t2, post_blur</i>) settings for each column are as follows. 1:(10, 5, 20, 20); 2:(10, 10, 50, 20); 3:(10, 10, 100, 30); 4:(15, 10, 100, 30).	167
6.1	Photos from <i>Deep Meditations: A brief history of almost everything in 60 minutes</i> at Sonar+D Barcelona, 2019	174
6.2	An example frame from a rendered video of a (<i>z</i> -sequence, video) pair. Each of the 28 tiles in this image, shows the image produced by decoding the same <i>z</i> -vector from 28 different snapshots of the same model where each snapshot is separated by 100 iterations. The snapshot iteration number is also included in each square panel for reference, although in a font perhaps too small for print.	182
6.3	A screenshot of editing <i>z</i> -sequence videos in Kdenlive with tiled snapshots.	185
6.4	An interpolated, dense <i>Z</i> sequence produced using spherical interpolation. Time flows left to right. Upon close inspection, one can notice vertical notches. These indicate discontinuities every time a keyframe is reached, and a new target keyframe is selected.	188

6.5 An interpolated, dense Z sequence produced using physical interpolation. Time flows left to right. The vertical notches that were visible in Fig. 6.4 are no longer an issue. 191

List of Algorithms

1	Hello World: Update loop	105
2	Learning to see: Training pipeline	135
3	Deep Meditations: Generation of random <i>short journeys</i>	184
4	Deep Meditations: <i>Conforming</i> the edit	186
5	Deep Meditations: Produce a dense trajectory Z using <i>spherical interpolation</i> .	188
6	Deep Meditations: Produce a dense trajectory Z using <i>physical interpolation</i> . .	192

Chapter 1

Introduction & Motivations



Figure 1.1: The hand stencils at the *Cuevas de las Manos* in Santa Cruz, Argentina, are thought to be over 10,000 years old. Image from Wikimedia Commons by User:Marianocecowski, licenced under CC BY-SA 3.0.

1.1 Background

Humanity's desire to build and use tools for artistic, creative expression dates further back than we often realise. Recent archaeological discoveries from just the last few decades, continue to radically question our notions of when this creative endeavor began.

The cave paintings of the *Altamira cave* in Spain, are thought to be some 36,000 years old (Pike et al., 2012). Flutes carved out of animal bones and mammoth ivory, such as those found in the *Geissenkloesterle caves* of southern Germany, are thought to be over 40,000 years old (Higham et al., 2012). The *Lion Man of the Hohlenstein Stadel* from the same region, an anthropomorphised lion-headed figurative sculpture carved out of mammoth ivory using flint cutting tools, is also thought to be almost 40,000 years old (Ulm, n.d.). Older examples such as the *Venus of Tan-tan* from Morocco, or the *Venus of Berekhat Ram* from the Golan Heights, are thought to be as old as up to 300,000 to 700,000 years old!¹

It is clear that even in the earliest stages of human, and potentially even pre-human history, we have sought out the most cutting edge technology available at the time, to create tools and instruments for artistic, creative expression.

Computers have been involved — even if only conceptually — in the creation of art and creative endeavours for as long as computers have existed. In her 1843 *Notes (On the Analytical Engine)*, the pioneering mathematician and computer programmer Lady Ada Lovelace foreshadowed the creative computational revolution we are living today (Fuegi & Francis, 2003):

“We may say most aptly, the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves.” — Ada Lovelace, 1843

While her collaborator Charles Babbage, designer of the *Analytical Engine*, was primarily focused on the number crunching abilities of the machine, Lovelace saw the potential of such apparatus to go further, and through symbolic manipulation, perform true, general purpose computing.

“Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.” — Ada Lovelace, 1843

Today, **Computational Art and Design (CAAD)**, the use of computation and algorithms in the production of artistic and creative works — be it images, video, music, sound, text, or 3D geometry — is a well established area of research and artistic inquiry. As our computers are becoming ‘smarter’, algorithmically generated media is becoming even more ubiquitous. The use of **Procedural Content Generation** in games is gaining popularity, and even launching whole new genres (Shaker et al., 2016). The tools available to designers are becoming more powerful, offering more capabilities, and automating laborious tasks. Algorithmic media production tools are now also now being presented directly to the masses through a plethora of ‘fun and creative’ mobile applications and ‘smart filters’ on social media such

¹Due to the sub-optimal conditions of these particular prehistoric artefacts, there is some controversy regarding their origins, and whether they were carved by human hands, or by natural geological processes.

as Instagram, SnapChat, and TikTok. The international publication *Art in America*, which typically concentrates on traditional contemporary art, has acknowledged the significance of computational art, dedicating an entire issue to it in January 2020 (Bailey, 2020; Caplan, 2020; Plummer-Fernandez, 2020).

Now, with the recent developments in **Machine Learning**, and particularly **Deep Learning**, as computational artists and designers, as researchers and developers of creative tools, we are entering an exciting and fruitful new era in this field.

Machine Learning (ML) is a field of research within **Artificial Intelligence (AI)**, that investigates how algorithms can improve their performance on various tasks by learning from experience. These algorithms offer tools, to help identify complex relationships and patterns in data. In doing so, they allow us to build systems that are capable of performing tasks that we do not explicitly know how to solve or formulate.

Deep Learning (DL) is a field of research within Machine Learning that investigates how algorithms can learn from vast amounts of high-dimensional, highly complex ‘raw’ data, such as images with millions of pixels, or sounds with thousands of samples per second.

ML, and even DL, have been active areas of research for many decades. In his 1948 lecture and accompanying essay *Intelligent Machinery, A Heretical Theory*, the renowned computer scientist Alan Turing described *B-Type Unorganized Machines*, an analog of the Artificial Neural Networks we use today, theoretical machines that *learn from experience* (Turing, 1948).

With significant developments in parallel computing architectures in recent years, combined with the availability of massive datasets assembled via the internet, and fuelled by the multi-billion dollar investments from The Surveillance Economy and The War on Terror, researchers have been able to develop and apply Deep Learning algorithms to highly complex, real-world problems. Today, Deep Learning permeates every aspect of our lives. These algorithms are in our pockets, organising our photos (Touvron et al., 2019), enabling our devices to speak (Shen et al., 2018), and understand what we say (Veton Kėpuska & Bohouta, 2017). They recommend films and music (S. Wang et al., 2014), perform medical image analysis (Litjens et al., 2017), predict protein folding structures (Yang et al., 2020), drive cars (Huval et al., 2015), and beat the world’s top players at games such as Go (Silver et al., 2018) and StarCraft II (Vinyals et al., 2019).

Very recently, we have also started seeing very impressive results in the application of Deep Learning algorithms to the production of artistic works and creative media. These algorithms can now produce photorealistic images (Karras et al., 2020), write poems and essays (Brown et al., 2020), collaborate on sketches (Ha & Eck, 2017) and generate music complete with lyrics and vocals (Dhariwal et al., 2020).

For the sake of simplicity, we will refer to the application of DL to the production of artistic works and creative media, as **Creative DL**. This includes both the technical DL research within this field, as well as the artistic and/or design practices and research.

Our research is situated at this intersection. And a major limitation that we observe in this area, is that the majority of the methods available offer very limited, if any, creative control to a human user. This is precisely the topic that our research investigates. Furthermore, we are not only interested in ensuring that a human user remains in creative control, but we want to ensure that this control occurs in a realtime, continuous manner such that the interaction is *performative* and *expressive*.

In other words, we investigate **Deep Learning models as an artistic medium for new modes of *performative, creative expression***.

To be more precise, our primary research question is:

How can we design and develop *Deep Visual Instruments*: realtime interactive generative systems that exploit and leverage the capabilities of state-of-the-art Deep Learning algorithms, while allowing *Meaningful Human Control*, in a *Real-time Continuous* manner?

1.2 Why Deep Learning?

If **Machine Learning (ML)** is the study of algorithms learning from *data*, then **Deep Learning (DL)** is the study of algorithms learning from **Big Data**, via ‘deep’ parametrisable computation graphs that learn hierarchies of ‘concepts’ (LeCun, 2014; I. Goodfellow et al., 2016).

The fact that DL algorithms require vast amounts of data is often one of the criticisms brought against DL. These algorithms do perform exceptionally well at many tasks, significantly outperforming *non-deep* methods by large margins (Krizhevsky et al., 2012; Hinton et al., 2012; L. Deng et al., 2013). But they require vast amounts of data to do so, often millions of training examples (J. Deng et al., 2009), or even billions (Mikolov et al., 2013). Since millions or billions of data points are not always available, learning from *few* examples, i.e. *one-shot learning* or *zero-shot learning*, is a growing area of research (Santoro et al., 2016; Vinyals et al., 2016; Rezende et al., 2016; Xian et al., 2018).

However, for the purposes of this thesis, the fact that DL algorithms require vast amounts of data is not a handicap, it is a *feature*.

Non-deep learning algorithms — such as logistic regression, support vector machines, shallow Neural Networks etc. — have a much harder time modelling highly complex, high-dimensional problems². For this reason, when working with such algorithms it is typically necessary to preprocess the data in order to reduce the number of dimensions. These hand-crafted, domain-specific features are then presented to the algorithm for modelling (LeCun, 2012). This process of *feature engineering* itself is often difficult, time-consuming and requires expertise (Ng, 2013). Furthermore, the performance of the model is highly dependent on the chosen hand-crafted representation (Bengio et al., 2013). While these feature engineering based approaches do prove useful in many domains, they can also provide inconsistent, unreliable and suboptimal results in more complex settings.

²We discuss the *expressive power* of Artificial Neural Networks in section 2.3: *Introduction to Deep Learning*

Addressing these issues is a major motivation that drives DL research. When designing a DL model, the feature engineering phase can be omitted. Instead, the model can be trained directly on the much higher dimensional ‘raw’ data. In effect, something akin to a *preprocessing pipeline* is *learnt* during training. However, to be able to model such complex, high-dimensional data, the model requires many parameters, often millions, sometimes even *hundreds of billions* (Brown et al., 2020). And with so many more model parameters to learn, this comes at the cost of more complex architectures and implementations, much higher computational requirements, and vastly larger datasets.

In the context of this thesis, we consider Deep Learning algorithms as **tools to extract useful information from a vast sea of humanly-unmanageable Big Data**. And it is precisely this aspect of Deep Learning that motivates our research. Our goal is to investigate ways in which we can navigate and explore *what* useful information a Deep Neural Network has learnt and extracted from the vast amounts of data. More specifically, we investigate how we can *meaningfully* use such a model in the production of *artistic and creative works*.

In other words, if for example a DNN is trained on a massive dataset of millions of images, how can we design and implement interactive generative systems that allows a person to use this model to generate new images, in a way that affords them *Meaningful Human Control* over the generated images, in a *Realtime Continuous manner*.

1.3 Meaningful Human Control

Meaningful Human Control is a term that we adopt from the *Autonomous Weapons Systems* literature (Scharre & Horowitz, 2015). There should be little doubt that our research in Creative DL is not as lethal as autonomous weapons, such that it would require security briefings at the United Nations³. Nevertheless, the term *Meaningful Human Control*, especially as we adapt it, perfectly captures the motivations behind our research.

In the context of autonomous weapons, Meaningful Human Control is defined with regards to a *threshold of human control that is considered necessary for the weapons system to be ethically, legally, operationally and diplomatically acceptable*. The specificities of that threshold in the context of autonomous weapons is not directly applicable to our research. In fact, in the context of our research we do not feel the need to define a *threshold* per se. Instead, we think of Meaningful Human Control as a *continuum* where we can have *more or less* (or *no*) Meaningful Human Control. This is analogous to how we think of *creativity*. Instead of trying to define a threshold which separates *creative* from *not creative*, we consider creativity to be a (multi-dimensional) continuum (Boden, 2004).

With that in mind, in this thesis we do not define a threshold for Meaningful Human Control. Instead, we investigate methods of interacting with DNNs for the production and manipulation of creative media, while always aiming to *maximise* the level of Meaningful Human Control.

³While it is not a focus of this thesis, developments in *DeepFakes* and politicized synthetic media is a very real and increasingly growing point of concern and consideration.

1.3.1 Pressing a button

Once again borrowing from the autonomous weapons literature: “*A human simply pressing a ‘fire’ button in response to indications from a computer, without cognitive clarity or awareness, is not sufficient to be considered ‘human control’ in a substantive sense.*” (Roff & Moyes, 2016). This statement is perfectly inline with our notion of Meaningful Human Control in the context of artistic and creative expression.

Given a generative system capable of producing ‘beautiful’ images, a human simply pressing a ‘generate’ button, which in turn produces a stunning image, is not within the scope of this thesis. While we acknowledge that this is a highly challenging and respectable area of study in itself — technically, artistically and philosophically (Boden, 1998) — and with many real world applications, our goal is not to create *autonomous media creation systems*⁴

This ‘generate’ button example may seem hypothetically extreme, but it is in fact an accurate representation of the dominant paradigm in the current Creative DL landscape. One of the most common workflows currently made available and employed in this area consists of training a generative DNN on some data, and then taking ‘random’ samples from the model — i.e. the programmatic equivalent of pressing a ‘generate’ button. In these cases, other than the act of *curation*, the person has no control over the media generated.

Curation of the *outputs* from the model is an act of *discrimination*, whereas in this thesis we are seeking to enhance the act of *creation*. Curation of the *inputs* to the model, i.e. the training data, does not influence an *individual* output, but the *entire space of possible outputs*. Particularly in Deep Learning — where the number of required training examples are often in the millions, and training a model can take weeks or even months — this does not provide a practical solution to the *fine level of control* that we seek in this thesis.

This is not to say that we do not value the act of *curation* as a creative and artistic activity. In fact, artists such as Helena Sarin⁵, Sofia Crespo⁶ and Anna Ridler⁷, to name just a few, continue to produce truly unique and impressive works that carry their own individual expressive signatures, often through painstakingly meticulous manual *curation* and *creation* of custom datasets. In other words, one can look at the images produced by these artists and immediately recognize them as their respective works. In this respect, it is clear that these artists do exercise very high levels of Meaningful Human Control over the images that they produce.

For this reason, it is important to distinguish that in this thesis we are investigating Meaningful Human Control *at the algorithmic level*. In other words, while curation and creation of custom datasets can be considered a creative act, we are interested in exploring *additional methods of Meaningful Human Control, at the algorithmic level*, that can enhance people’s experience of the *creation process*, to offer more capabilities than is currently available.

⁴We briefly discuss a field of research focused on this question, in subsection 2.2.8: *Computational Creativity*.

⁵<https://www.instagram.com/helena.sarin/>

⁶<https://sofiacrespo.com>

⁷<https://annaridler.com>

1.3.2 ‘Random’ faders

Instead of a single ‘generate’ button, we can imagine a scenario where a human has access to a number of adjustable options, such as a user interface with rotary knobs or sliding faders. If the person is unable to comprehend the connection between the faders and the output of the generative system, we do not consider this to be Meaningful Human Control. Taking this to the extreme, we can think of examples whereby the faders either i) don’t actually have any effect on the system at all, or ii) are simply seeds for a random number generator used in some way by the generative system. From our point of view, with respect to Meaningful Human Control, these two cases are functionally equivalent to the case where the faders do in fact affect the output but a human is unable to comprehend the connection, and use them in a meaningful way.

This may again seem like a hypothetical example. However, it is grounded in reality. Deep generative models learn compact, *latent representations* of the data that they are trained on. These representations are high-dimensional vectors, and the individual components are typically not related in any way to any single humanly-interpretable real-world characteristic of the data. For this reason, from a Meaningful Human Control point of view, manually manipulating the components of these vectors is functionally equivalent to modifying the seeds of a random number generator. In this thesis we look for alternatives to direct manipulation of latent vector components in this manner. Interestingly, this is currently a growing area of research, and orthogonal to our work, there are a number of approaches that are being developed such as *disentangling* these latent representations (Chen et al., 2016; Higgins et al., 2017), or discovering human-interpretable *semantic vectors* (Simon, 2019; Karras et al., 2019, 2020; Härkönen et al., 2020). We discuss these in section 2.1: *Generative models*.

1.3.3 Necessary and sufficient conditions

Based on the discussion above, we identify three conditions that forms the basis of Meaningful Human Control in the context of this thesis:

Intent: For a generative system to allow Meaningful Human Control, a key requirement is that the system, and the *interaction with the system*, is able to incorporate and translate a human’s *intent* into the output that it produces. In the case of our research in artistic and creative media — which, unlike autonomous weapons, is not lethal — it is not essential that a person has a particular goal to begin with. In fact, it is very common in creative explorations to embark on *goal-less*, curiosity-driven meanderings, and these can often turn out to be very fruitful (Secretan et al., 2008; Stanley & Lehman, 2015). However, as a person interacts with the system in such an exploratory manner, it is not uncommon that a goal, initially perhaps just a vague direction, might begin to appear and crystallize. As this happens, an interaction with Meaningful Human Control should be able to both *deliver*, and *guide* that intent.

Predictability: For the above to be possible, we also consider it crucial that the generative system, and particularly the interaction with the system, is *predictable*. Once again, in the case of our non-lethal research in artistic and creative media, this predictability need not be

apparent *instantly* or *absolutely*. A person can *eventually* start to build an understanding of the system to the extent at which they can predict the outcome of their actions and feel that they have some level of control. If the system and interaction is too unpredictable for a person to build an understanding of and use in such a manner, then we do not consider the system to allow Meaningful Human Control.

Accountability and expression: When a generative system has sufficient *predictability*, and a human’s *intent* is successfully incorporated and translated into a particular outcome, we can consider the human to be *accountable* for that outcome, since it is their *informed, conscious decisions and actions* which led to this *specific, unique* outcome. On the other hand, looking at it not from the *system’s perspective*, but the *human’s*, we consider the generative system to allow Meaningful Human Control if a human is able to *creatively express themselves* through the system. In other words, if the outcome represents what the person sought to produce, and it contains their *personal, expressive signature*.

In the context of artistic and creative works, these qualities are very difficult to systematically quantify. We discuss our approach to evaluation in subsection 1.7.2: *Research methods and evaluation*, and we reflect on our approach in more detail in section 7.2: *Research methodology*.

1.4 Visual instruments: Realtime Continuous Control

1.4.1 Realtime Continuous Control

In the previous section, as an integral point of focus for this thesis, we presented the concept of *Meaningful Human Control*, a term that we adopt from the autonomous weapons literature. In this section, as an additional point of focus, we present **Realtime Continuous Control**, a term that we adopt from the *cybernetics and control theory* literature (Wiener, 1948). In the context of this thesis we envisage a *closed-loop* between a human, and an interactive generative system which incorporates a DNN. The human *continuously monitors* the outputs of the generative system in realtime, and *guides it towards desirable outcomes*. In the following paragraphs, we explain why we see this as a useful model of interaction.

1.4.2 Visual instruments

One analogy that we use to frame our research when thinking about interactive generative systems, is the visual equivalent of a musical instrument. This is to say that, one can interact with the system in a Realtime Continuous manner, analogous to how one might interact with a musical instrument, such as a piano. We think of these **visual instruments** in the lineage of Louis-Bertrand Castel’s 18th century *Ocular harpsichord* (Castel, 1740). This was a modified harpsichord that on each keypress, would project light of different colours onto a large surface, allowing a performer to simultaneously perform music and colour. In this lineage, we also think of electronic video synthesizers such as the *Paik-Abe Video Synthesizer* built by Shuya Abe and Video Art pioneer Nam June Paik in 1969–1971, and the *Rutt-Etra Video Synthesizer* built by Steve Rutt and Bill Etra in 1972 (Collopy, 2014).

1.4.3 Realtime performative interaction

These instruments are designed for **realtime performance**. However, in using the word *performance*, it is important to underline that we do not necessarily mean a *live performance in front of an audience*. Instead, we simply mean that the media is *created live, in realtime, with continuous control, in a performative manner*. This may be in front of a live audience. Or it may be in a studio, recorded and later presented as a non-interactive, non-realtime, traditional film or animation.

Creating media in this manner, interacting with a generative system with Realtime Continuous Control, allows a user to experiment, explore, search for and find configurations that produce desirable, and potentially *more novel and previously unimaginable outcomes*.

1.4.4 Goal-less exploration

We believe Realtime Continuous Control can potentially produce *more novel and previously unimaginable outcomes*, because such an interaction allows a person to freely explore a massive space of possibilities. Initially, the user may not have a clear idea of what it is they would like to create, so they may embark on a **goal-less**, purely inquisitive, creative exploration. During this exploration, they may perform investigatory interactions with the system, probing, and observing the results, to help build an understanding of the system's creative capabilities. Continuously interacting with the system and observing the results with *immediate feedback*, can help the user learn how to *more meaningfully control* the system. Goal-less explorations such as this are known to show great potential in producing novel, unexpected discoveries which can spark new ideas and encourage new explorations into new directions (Secretan et al., 2008; Stanley & Lehman, 2015). Furthermore, as the user discovers interesting new territories in the possibility-space, they may begin to visualise a clearer goal. This vision may not be very concrete to begin with. But with time, as they explore more, while **continuously monitoring** the results of their interactions, they may guide the system towards desirable outcomes.

Many early video artists used these early video synthesizers in this way to create seminal works, and Paik himself writes in his manifesto *Versatile Video Synthesizer*, while demonstrating the capabilities of his machine (Source: Kat. Nam June Paik, *Videa 'n Videology 1959–1973*, Emerson Museum of Art, Syracuse, New York, 1974 p.55 (Medienkunstnetz.de, n.d.)):

This will enable us to shape the TV screen canvas
as precisely as Leonardo
as freely as Picasso
as colorfully as Renoir
as profoundly as Mondrian
as violently as Pollock and
as lyrically as Jasper Johns.

In this respect, we are generally not interested in systems that *can* provide Meaningful Human Control, but *not* in this Realtime Continuous manner that we describe above. For

example, the Non Linear Video Editing (NLVE) software *Adobe Premiere* does provide Meaningful Human Control. However, it cannot be considered a *realtime performance instrument*⁸. A more suitable example that we can pull from the visual domain, is *drawing or painting*. These activities, similar to playing a musical instrument, do provide Realtime Continuous Control in a performative and expressive manner.

Again, this is not to say that we do not value the effectiveness of non-realtime, non-continuous modes of interaction in the production of artistic and creative works. Such modes of interaction are quite common in many typical, established creative workflows. As we have already mentioned, traditional NLVE is an example of this. In fact, in chapter 6: *Deep Meditations: Latent storytelling* we present a study based on this exact workflow. However, in this thesis, we primarily focus on *Realtime Continuous Control* because we believe it to be a very under-explored, yet very valuable mode of interaction.

1.4.5 Flow

A key theme that connects the activities that we mention above and use as metaphors to base our research on, is *flow*.

Flow is a state of mind, and mode of being, where a person is fully immersed in an activity. Their sense of time is distorted, they are aware of nothing but the act that they are engaged in, and the activity itself becomes autotelic (Mihaly Csikszentmihalyi, 1996). Colloquially known as ‘being in the zone’, many activities can induce a state of flow. These include the activities that we have already mentioned, playing musical instruments, painting and drawing; as well as many others such as writing, dancing, playing games or sports, and cooking etc. It is well documented that being in a state of flow correlates with *enhanced creativity, enjoyment, focus, motivation and productivity* (Csikszentmihalyi et al., 2005).

In our research, we do not focus on specifically inducing and testing for states of flow. We also do not claim that the systems which we create *do* induce states of flow. Instead, we draw inspiration from this area of research, and from the activities which are known to induce flow, and we use them as guidelines.

One of the key requirements for being in a state of flow is a feeling of **agency and control** over the activity being performed. In other words, flow requires *Meaningful Human Control*.

Another requirement for being in a state of flow, is **immediate feedback**. This establishes the creative feedback loop between the person, and the environment in which the activity is taking place. Actions taken by the person have an immediate effect and produce a new state or outcome. This outcome sparks new ideas and feelings in the person, and they are able to respond in realtime to what they are performing or creating. In other words, this is the **closed-loop** of the Realtime Continuous Control system (Wiener, 1948) that we have just discussed.

All of the studies that we present in this thesis (with the exception of one, which we will discuss the motivations for in chapter 6: *Deep Meditations: Latent storytelling*), we demonstrate with *realtime software systems* that we have developed, that run with a minimum framerate of

⁸We do not claim that it is *impossible* to use Premiere as a realtime performance instrument. In fact, we would not be surprised if there exists a niche community of experimental video artists who perform live entirely using Premiere. However, the software is clearly not designed for realtime performance.

15 frames-per-second on affordable (high-end consumer) hardware. A user can interact with our software via a number of different modalities at any point in time while the software is running. The latency between user input and visible results is usually in the order of one or two frames. In other words, the latency across all of the software systems that we have developed is in the range 16–130 ms. We consider this to be more than sufficient Realtime Continuous Control.

1.5 Creative DL \times Meaningful & Realtime Continuous Control

1.5.1 The State

At the start of our research in 2014, Deep Learning algorithms had already started to demonstrate their superior performance — compared to other methods, and often to humans — in many complex *classification* tasks in fields such as speech recognition (Hinton et al., 2012; L. Deng et al., 2013), natural language processing (Collobert et al., 2011), handwriting recognition (Pham et al., 2014), image classification (Krizhevsky et al., 2012), image scene labelling (Couprie et al., 2013), spam filtering (Guzella & Caminhas, 2009), and many others (Schmidhuber, 2015).

However, the area that we call **Creative DL**, i.e. the application of DL to the *production of artistic works and creative media*, was very much still in its infancy. Research in Creative DL was incredibly sparse and relatively primitive in terms of quality, and far from being usable in any production environment. Nevertheless, these early studies did demonstrate incredible potential in fields such as MIDI music composition (Boulanger-Lewandowski et al., 2012; Nayebi & Vitelli, 2015; Sturm, 2015), text generation (Sutskever et al., 2011; Sutskever, 2013), drawing (Graves, 2013; Ha, 2015) and image generation (Gregor et al., 2015; Nguyen et al., 2015; Gatys et al., 2015a, 2015b; Mordvintsev et al., 2015; Radford et al., 2015; Nayebi & Vitelli, 2015).

Today, just five years later, the resolution of the images produced by DNNs are orders of magnitude higher (Karras et al., 2020); instead of MIDI music composition, DNNs are generating music as raw audio samples, complete with lyrics and vocals (Dhariwal et al., 2020). Creative DL, is now a very active area of research with strong interest and support from industry giants such as Google, Facebook, Snap, Microsoft, Adobe, Autodesk, OpenAI, Nvidia, Unity3D, Unreal Engine, Blizzard Entertainment and many more.

1.5.2 The Problem

As we have already mentioned, one theme that connects much of this research, is that they offer a human user very limited control over the outcomes generated. Definitely not at a level that we would consider as satisfactory levels of Meaningful Human Control. And absolutely not in a realtime, interactive manner with continuous control. As we have explained in previous sections, we believe this to be a very under-explored, yet vital area of research.

1.5.3 The Reason

The reason that this area is so under-explored, we believe is due to the fact that many of these generative DL algorithms were themselves in their infancy. For this reason, the majority of

technical research in Creative DL was not focused on Meaningful Human Control, but instead on trying to improve the performance, reliability and stability of the learning algorithms themselves. Only in the last few years — as these algorithms have matured and begun to demonstrate improved performance, reliability and stability — more research has started to shift attention towards investigating ways of allowing Meaningful Human Control over the systems, and this is now a rapidly growing area of research (Isola et al., 2016; Zhu et al., 2017; Ha & Eck, 2017; Karras et al., 2017; Park et al., 2019; Karras et al., 2019; Simon, 2019; Bau et al., 2019; Karras et al., 2020; Härkönen et al., 2020; Jiang et al., 2020; Broad et al., 2020). We discuss this timeline in more detail in subsection 2.2.7: *Creative Deep Learning — from a cultural perspective*

Our research has been based on the expectation that these algorithms would be rapidly improved, optimized and stabilized, with huge engineering efforts and investments from the likes of Google, Facebook, Microsoft, Adobe, Autodesk, OpenAI and Nvidia. And indeed this expectation has been, and is continuing to be, met. For this reason, our research has been focused from the start, on how these new and emerging DL algorithms and techniques can be used in the production of artistic and creative works, with both *Meaningful Human Control*, and *Realtime Continuous Control*, to allow for *performative, creative* expression.

1.6 Conclusion: why is this important

In summary, we believe:

- DL algorithms that learn hierarchies of features and semantic latent representations directly from high-dimensional raw data, without the need for hand-crafted feature engineering, have tremendous potential when applied to fields involving the production of artistic and creative works.
- Harnessing the capabilities of DL in the creative industries, does not only automate and optimise previously very tedious tasks (such as rotoscoping or image segmentation), but it opens up whole new avenues with regards to what is possible.
- It is incredibly valuable to develop methods that grant people — artists, designers, and the general public — Meaningful Human Control over these generative algorithms, such that people are able to express themselves, and execute their vision via these systems.
- These methods can be incorporated into specialist design applications across a wide range of creative industries including video games, movies, music, graphic design, architecture, industrial design, theatre, dance, publishing and many more. They can also be incorporated into consumer-facing products, such as ‘creative and fun’ desktop or mobile applications, or even social media ‘smart filters’.
- This potential is mirrored by the amount of recent growing interest from industry giants such as Google, Facebook, Microsoft, Adobe, Autodesk, OpenAI, Nvidia, Snap, Unity3D, and many others.

- A generative system incorporating a DNN has the potential to offer an incredibly large space of possibilities. Exploring such a vast space presents many challenges.
- It is incredibly valuable that a person has the ability to freely explore such a massive space, so that they may embark on an goal-less, purely inquisitive and creative exploration, to build an understanding of the extents of such a system’s creative capacity (Secretan et al., 2008; Stanley & Lehman, 2015). With time, they may begin to establish an idea, or a vision. This may be vague to begin with, a very large ‘destination’ so to speak. But with time, as they explore further, this vision may become clearer as they seek to hone in on a specific target.
- *Meaningful Human Control*, combined with *Realtime Continuous Control*, is essential to allow a person, optimal balance between *exploration* of the yet unknown spaces of such a complex, deep generative model, vs *exploitation* of their own knowledge, vision and intent.

1.7 Research

1.7.1 Research summary

Our research is situated at the intersection between state-of-the-art Deep Learning algorithms; Computational Art and Design; and Artistic, Expressive Human-Computer Interaction. Within this interection, we are interested in exploring Deep Learning models as an artistic medium for new modes of performative, creative expression. Our primary research question is:

How can we design and develop *Deep Visual Instruments*: realtime interactive generative systems that exploit and leverage the capabilities of state-of-the-art Deep Learning algorithms, while allowing *Meaningful Human Control*, in a *Real-time Continuous* manner?

The questions and approaches that we present could be adapted to various different domains. However, in this thesis we choose to focus on *images* and *videos* (i.e. sequences of images). We choose images as a starting point for a number of reasons. Due the pervasive nature of images as a medium, there is a plethora of image data on the internet that can be collected and organised into datasets. As a result of this, DL research in image based DNNs are more mature in comparison to other ‘rich’ mediums such as sound or 3D geometry.

We encounter a number of challenges when addressing this topic:

- Given that most current DL architectures offer very limited control over the outputs generated, how can we design points of interaction into the architectures that allow *Meaningful Human Control*?
- Given that DL algorithms are notoriously compute-intensive, how can we design systems that are able to run on affordable hardware, in *realtime*, with *continuous* control?
- Having overcome the previous two challenges, how can we design interactions that allow or enhance both *Meaningful Human Control*, in a *Realtime Continuous* manner?

- What is the impact of, and unique considerations for, using different learning methods and architectures, such as Monte Carlo Tree Search (MCTS), Long Short Term Memory Recurrent Neural Networks (LSTM RNN), Variational AutoEncoders (VAE) and variants of conditional and unconditional Deep Convolutional Generative Adversarial Networks (DCGAN)?
- And finally, what are the necessary characteristics of Meaningful Human Control and Realtime Continuous Control?

1.7.2 Research methods and evaluation

As we have explained in the previous sections, we believe that DNNs have great potential when used for the production of artistic and creative works, especially when allowing a user Meaningful Human Control, in a Realtime Continuous manner. We also believe, that this is currently a very under-explored area of research.

The goal of our research is not only to invent one or more *specific methods* that achieve this. Instead, we wish to also demonstrate the potential, and the importance of this particular field, and we wish to encourage more general research in this direction.

For this reason, we present five different studies in this thesis, each taking a very different approach to this problem. For each of these studies we have developed a software tool that demonstrates the method that we propose in the corresponding study. Using these software tools that we have developed, we also produce and publicly share a number of artworks — in the form of video outputs and installations — to further demonstrate the capabilities and potential of the approaches that we investigate, hopefully inspiring more research in this field.

In each study, as opposed to extensively developing each software tool to the extent where they can be used by the general public, we instead prefer to invest our time across different studies, to investigate different methods. We develop each method and software enough to demonstrate its potential to the extent that it can ideally provide a foundation for future research. We believe that this multi-pronged approach may potentially have more impact and may encourage more research in this field.

It's important to note that we do not conduct user studies, and we do not make claims regarding the generalizability of the interfaces that we develop to a wider audience. We also do not research or make claims regarding which specific interaction models are more or less effective at granting Meaningful Human Control or inducing a state of flow, if they were to be deployed to a general audience. Instead, as we will shortly explain in more detail, we use the methods and software tools that we develop in this thesis to create a number of artworks, and we use our own subjective experience of creating these artworks as a way of reflecting upon and evaluating the methods and software tools that we develop. We choose this route because again we believe that focusing on a higher number of personal demonstrations — as opposed to fewer, but more thoroughly user-tested studies — has a larger potential for impact with regards to inviting a more diverse range of artists, creatives and researchers into thinking about Meaningful Human Control and Realtime Continuous Control within Creative DL.

We research methods that allow the introduction of *any* forms of realtime interaction and control, into currently *non-realtime, non-interactive, non-controllable* processes. While doing

so, we take Meaningful Human Control and Realtime Continuous Control as a framework and strong guiding points.

We evaluate our methods by assessing i) if they introduce *any* form of control at all, to previously *uncontrollable* workflows; ii) if they introduce any form of *Realtime Continuous* control, to previously *non-realtime, uncontrollable* workflows; iii) if they provide any *Meaningful Human Control*, subjectively judged via our own ability to utilize these methods in producing creative works that carry our *intent* and *personal expressive signature*; and iv) online and offline response to the proposed methods and related works, and if and how they influence the works of other practitioners and researchers. We primarily assess this through social media and mainstream publications and coverage.

Furthermore, as a media artist who has focused on developing realtime interactive computational systems for creative expression for almost two decades, we leverage our own experience in this field to assess the methods that we present in this thesis. Throughout this research, we create a number of artworks using the methods that we develop, and we use these artworks as a test-bed for evaluating the methods. It is in support of creating these artworks, that we are able to assess, refine and reflect on the systems and methods that we develop and present in this thesis. For the purposes of this thesis however, these creative artefacts themselves are not the basis of contribution to knowledge, and for that reason our research should not be considered *practice-based* research (Candy, 2006). Instead, the artworks that we produce during this research have been independently peer-reviewed (within various exhibitions and publications, and we share these details in the following sub-section, and throughout the thesis where relevant). And the artworks act as demonstrations of the potential efficacy of the approaches that we present, and the experimental subject matter which we use when testing, refining and reflecting upon the methods that we develop.

This research methodology, especially in conjunction with the fact that we have not conducted user-studies, may be considered a limitation of our research. Indeed more work — particularly around user-testing and usability refining — would be needed before the methods that we propose in this thesis could be used in end-user facing, production-ready environments. And for this reason we do not make claims regarding the generalizability of our methods to other users — although casual observations of the general public interacting with our interactive installations does appear to be promising. Instead, in this thesis we have focused on demonstrating the potential of this new medium from multiple different angles, and the vast space of possibilities that such approaches can offer. We hope that this can inspire others to investigate similar approaches, either building directly upon our own work and methods, or even exploring tangentially related methods. We reflect upon our research methodology in more detail in section 7.2: *Research methodology*.

1.8 Summary of contributions and impact

In this thesis, we present five studies, which each take a different approach to the topic of both *Meaningful Human Control* and *Realtime Continuous Control* over generative systems that incorporate DNNs. These studies are:

- An agent-based, interactive, collaborative, generative sketching application using MCTS and discriminative CNNs (section 3.2)
- A system to gesturally conduct the generation of text in different styles using an ensemble of LSTM RNNs (section 3.3)
- Hello World: A performative tool that allows for the manipulation of hyperparameters in realtime while a Convolutional VAE trains on a live camera feed (chapter 4)
- Learning to see: A pix2pix based live camera feed transformation software that allows for digital puppetry and augmented drawing (chapter 5)
- Deep Meditations: A method that allows for long-form story telling within a generative model’s latent space with meaningful control over the narrative (chapter 6)

In addition, we have produced and publicly shared a number of videos and artworks using the tools that we developed as part of our research. We have shared these works online, on social media, and in galleries and inter-disciplinary, academic and non-academic festivals and conferences. In publicly sharing these outputs, we hope to demonstrate the great potential and possibilities that DNNs have to offer in the production of artistic and creative works, especially when *Realtime Continuous Meaningful Human Control* is provided. And in turn, we hope to encourage more conversations and research in this field, which we believe is currently very under-explored.

A selection of contributions and impact that we have made are listed below.

- For each of the five studies that we discuss in this thesis, we have presented either an accompanying paper, a live demonstration, or both, at a conference or workshop. These include an Art paper at SIGGRAPH (Akten et al., 2019), and three papers and a live demonstration at *Neural Information Processing Systems* (NeurIPS) and associated workshops (Akten & Grierson, 2016b, 2016a; Akten et al., 2018).
- The software and interactive installation that we present in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, has been included in MIT’s Open Documentary Lab *docubase*⁹.
- One of the video demonstrations that we created and discuss in chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, was shown during Nvidia CEO Jensen Huang’s keynote at GTC (GPU Technology Conference) 2019¹⁰, with the voiceover “[AI is] inventing new ways to bring out the creative genius in us all”.
- Established artists such as Scott Eaton and Patrick Tresset have very successfully incorporated workflows similar to what we demonstrate in chapter 5: *Learning to see: Digital puppetry through realtime video transformation* into their own artistic practice.
- We used the techniques and software that we developed in chapter 6: *Deep Meditations: Latent storytelling*, in a collaboration with the renowned electronic musician *Max Cooper*

⁹<https://docubase.mit.edu/project/learning-to-see>

¹⁰Nvidia GTC 2019 Keynote: <https://www.youtube.com/watch?v=Z2XlNfCtwlI&t=32>

for his seminal audio-visual performance and tour *Yearning For The Infinite* (Cooper & Akten, 2019). The piece we made for the performance, *Morphosis*, was also released online as a music video in 2020 (Akten & Cooper, 2020).

1.8.1 Press

Many of these works that we have shared on social media, has spread widely, both online and offline. We consider this to be an indication to the value of the opportunities granted by Realtime Continuous Meaningful Human Control with Creative DL, and the general interest in the field. These works and our research, have been featured on influential mainstream blogs and publications such as:

- Dazed Digital, 2020¹¹
- Clash, 2019¹²
- Art in the Digital Age, 2019¹³
- Artnome, 2018¹⁴
- Hyundai Art and Technology, 2018¹⁵
- Digital Trends, 2018¹⁶
- Boingboing, 2018¹⁷
- CreativeApplications.net, 2018¹⁸
- Flabber, 2018¹⁹
- Tech onliner, 2018²⁰
- popmech.ru, 2018²¹
- Gizmodo, 2018²²
- Prosthetic Knowledge, 2018²³
- We Make Money Not Art, 2018²⁴

1.8.2 Invited presentations and panels

We have had the opportunity to present our research as an invited keynote speaker or panellist, at a number of inter-disciplinary conferences. These include

- Ars Electronica, 2020 (online)²⁵

¹¹Deep Meditations on Dazed Digital

¹²<https://www.clashmusic.com/news/beyond-the-data-memo-akten-interviewed>

¹³http://artinthedigitalage.net/en/1902EN_plu.html

¹⁴<https://www.artnome.com/news/2018/12/13/machine-learning-art-an-interview-with-memo-akten>

¹⁵Hyundai Art and Technology <https://www.youtube.com/watch?v=1QIOoEID8kk>

¹⁶<https://www.digitaltrends.com/cool-tech/neural-net-rebuilds-reality>

¹⁷<https://boingboing.net/2018/03/21/watch-neural-networks-see-only.html>

¹⁸<https://www.creativeapplications.net/openframeworks/learning-to-see-making-deep-neural-network-predictions-on-live-camera-input>

¹⁹<http://www.flabber.nl/artikel/wat-gebeurt-er-als-je-een-videoalgoritme-alleen-maar-beeld-van-golven-vuur-of-bloemen-voert>

²⁰<https://tech.onliner.by/2018/03/20/elektroovcy>

²¹<https://www.popmech.ru/technologies/415152-kak-obmanut-iskusstvennyy-intellekt>

²²<https://gizmodo.com/trippy-magic-happens-when-ai-only-knows-about-flowers-1823900244>

²³<http://prostheticknowledge.tumblr.com/post/172012841001/gloomy-sunday-latest-addition-to-memo-aktens>

²⁴<https://we-make-money-not-art.com/doclab-exhibition-asks-are-robots-imitating-us-or-are-we-imitating-robots>

²⁵<https://ars.electronica.art/keplersgardens/en/returning-the-gaze-panel-1>

- Kikk Festival, 2019 (Namur, BE)²⁶
- TodaysArt, 2019 (The Hague, NL)²⁷
- AI in the Arts and Design panel, ACM Siggraph, 2019 (Los Angeles, CA, USA)²⁸
- Gray Area Festival, 2019 (San Francisco, CA, USA)²⁹
- Art and Artificial Intelligence, ZKM Center for Art and Media, 2019 (Karlsruhe, DE)³⁰
- Symposium of Interdisciplinary Artificial Intelligence Studies, Yeditepe University, 2019 (Istanbul, TR)³¹
- Cybernetic Consciousness Symposium, Itaú Cultural, 2019 (Sao Paulo, BR)³²
- Art Innovation Symposium, Kyoto University, 2019 (Kyoto, JP)³³
- TOCA ME Design Conference, 2019 (Munich DE)³⁴
- Muovo Conference on Motion Design, 2019 (Prague, CZ)³⁵
- Art Machines: International Symposium on Computational Media Art, Hong Kong City University, 2019 (HK)³⁶
- Innovative City Forum, Roppongi Hills Mori Tower, 2018 (Tokyo, JP)³⁷
- Smart cities and urban tech conference, Strelka Institute, 2018 (Saint Petersburg, RU)³⁸
- Workshop on Human-Computer Collaboration in Embodied Interaction, IRCAM, 2018 (Paris, FR)³⁹
- i-Docs, 2018 (Bristol, UK)⁴⁰
- Ars Electronica, 2017 (Linz, AT)⁴¹
- Sonar+D, 2017 (Barcelona, ES)⁴²
- DocLab, International Documentary Film Festival (Amsterdam, NL)⁴³

1.8.3 Public showings

The work that we have produced using the systems developed in this thesis, have also been shown in galleries, museums and cultural institutions such as

- Haus der Kunst, 2020 (Munich, DE),
- UCCA Center for Contemporary Art, 2020 (Beijing, CN)
- QUAD Gallery, 2020 (Derby, UK)
- Honor Fraser Gallery, 2020 (Los Angeles, CA, US)

²⁶<https://www.kikk.be/2019/en/home>

²⁷<https://todaysart.org/festivals/todaysart-2019>

²⁸https://s2019.siggraph.org/presentation/?id=bof_144&sess=sess370

²⁹<https://grayareafestival.io>

³⁰<https://zkm.de/en/event/2019/05/art-and-artificial-intelligence>

³¹<http://siais.yeditepe.edu.tr>

³²<https://www.itaucultural.org.br/conscienciacybernetica/2019>

³³<http://art.gsais.kyoto-u.ac.jp/index-en.html>

³⁴<http://www.toca-me.com>

³⁵<https://mouvo.cz>

³⁶<https://www.cityu.edu.hk/iscma>

³⁷<https://icf.academyhills.com/2018/en>

³⁸<https://inthecity.strelka.com/en>

³⁹<http://mim.ircam.fr/hamac>

⁴⁰<https://idocs2018.dcrc.org.uk>

⁴¹<https://ars.electronica.art/ai/en/symposium>

⁴²<https://sonarplusd.com/en/programs/barcelona-2017/areas/talks/the-role-of-artists-in-the-ai-revolution>

⁴³<https://www.idfa.nl/en>

- Kellen Gallery, 2020 (New York, US)
- SUPERCOLLIDER Gallery, 2020 (Los Angeles, CA, US)
- Centro de Arte y Creación Industrial, 2019 (Gijón, ES)
- Ars Electronica Centre, 2019 (Linz, AT)
- Mori Art Museum, 2019 (Tokyo, JP)
- Kate Vass Gallery, 2019 (Zurich, CH)
- Sonar+D Festival, 2019 (Barcelona, ES)
- Itaú Cultural, 2019 (Sao Paulo, BR)
- The Barbican, 2019 (London, UK)
- Kenninji Temple, 2019 (Kyoto, JP)
- Hatcham Church Gallery, 2018 (London UK)
- Moscow Museum of Modern Art, 2018 (Moscow, RU)
- Grand Palais, 2018 (Paris, FR)
- Astana Contemporary Art Centre, 2017 (Astana, KZ)
- Ars Electronica Festival, 2017 (Linz, AT)
- International Documentary Film Festival Amsterdam / IDFA (Amsterdam, NL)

1.8.4 Opensource software

We have developed and shared a number of open-source tools that we summarise below.

ofxMSAmcts (2015)

ofxMSAmcts (Akten, 2015) is an open-source C++ implementation of *Monte Carlo Tree Search* (MCTS) (Browne et al., 2012) that we developed as an addon for the very popular C++ creative development toolkit *openFrameworks* (Lieberman et al., 2016). *OpenFrameworks* has a large and very diverse user-base of artists, designers and creative developers, with roughly 250,000 visits per month to its website. Many of its users actively engage with new technologies and develop open-source *addons*, allowing the community to easily integrate third party libraries and devices. Examples of these addons include support for various hardware devices such as Microsoft Kinect, LeapMotion, Oculus Rift, DMX lighting control, ILDA laser control, quadrotor drones, robot arms, electroencephalography (EEG) brain activity readers, and video capture cards, as well as numerous software libraries for computer vision, networking, physics simulations, 2D and 3D graphics, audio synthesis and many more (*ofxAddons*, 2015).

Developing our MCTS addon for *openFrameworks*, enables us to instantly access the vast range of other libraries and tools developed for the *openFrameworks* ecosystem, including our own *ofxMSATensorFlow* (Akten, 2016). Our C++ library is general purpose, templated and supports a variable number of agents, and a variable number of actions per step. We developed and released a simple tic-tac-toe game to demonstrate how to use the library, and we use this library in our research in section 3.2: *Collaborative generative sketching with MCTS and CNNs*. The github repository currently has 43 stars and 15 forks.

ofxMSATensorFlow (2016)

There are many open-source Machine Learning libraries available, and many new ones being released frequently. Google’s *Tensorflow* library (Abadi & Others, 2015) was made public in late 2015 and immediately gained a lot of popularity. While these libraries are great for focused Deep Learning research, they are often difficult to integrate into different workflows and a wider ecosystem of tools. For this reason we integrated Tensorflow into openFrameworks.

ofxMSATensorFlow (Akten, 2016) is an open-source Tensorflow addon for openFrameworks that we developed, allowing Tensorflow to run seamlessly within this ecosystem of tools.

We developed and released a number of examples to demonstrate functionality on standard Machine Learning tasks as well as recent Deep Learning developments including classifying MNIST digits (LeCun & Cortes, 2010) and ImageNet images (J. Deng et al., 2009) with CNNs (Szegedy, Liu, et al., 2015), handwriting generation with LSTM RMDNs (Graves, 2013), text generation with character based LSTM RNNs (Graves, 2013), and ‘pix2pix’ Image-to-Image Translation with Conditional Adversarial Nets (Isola et al., 2016). The github repository currently has 445 stars and 92 forks.

webcam-pix2pix-tensorflow (2017)

In chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we present a realtime video processing system that can transform a live camera feed with the aesthetic qualities of a large image based dataset. Our system allows for the manipulation of a number of parameters in realtime via a GUI, to further transform the image. We have released a simplified version of this software as open-source software on github (Akten, 2017). This version contains a subset of the filters that we mention in this thesis, and lacks many of the advanced non-DL related options such as advanced screen layout options, changing and loading models at runtime, midi controller support etc. In simplifying the functionality and code, we hope that it will be more informative and educational, and can act as a base for others to build upon in the future. The github repository currently has 311 stars and 61 forks.

py-msa-kdenlive (2018)

In chapter 6: *Deep Meditations: Latent storytelling*, we take a typical Non Linear Video Editing (NLVE) workflow as a base, and we present a method that allows for long-form story telling within a generative model’s latent space with meaningful control over the narrative. In order to achieve this, we develop an open-source python script which can load and parse a project file belonging to the open-source *Kdenlive* NLVE software for the GNU/Linux platform. Our script can then *conform* the edit contained in the project, onto corresponding numpy arrays. We discuss this process in much more detail in the chapter linked above. This open-source script can be found on github (Akten, 2018).

1.9 Thesis outline

The remainder of this thesis is as follows.

In the following chapter, chapter 2: *Background*, we trace the multiple histories that span Computational Art and Design; Artistic, Expressive Human-Computer Interaction; and Deep Learning; until they converge and merge in the current day. We discuss major developments in these fields, especially those within the context of Creative DL, Meaningful Human Control, and Realtime Continuous Control. We also present the key concepts, terminology and technical foundations relating to Deep Learning, that we will be building upon and referencing in this thesis.

The majority of our research deals with pixel based raster images. However, since these are incredibly high-dimensional and computationally expensive, we choose to start with a number of lower dimensional studies. We present these two studies in chapter 3: *Realtime sequence generation with continuous control*. In this chapter, we present an agent-based, interactive, collaborative, generative sketching application using MCTS and a discriminative CNN . We also present a system to gesturally conduct the generation of text in different styles using an ensemble of LSTM RNNs.

The subsequent studies in this thesis focus on a number of different image based *generative models*.

In chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, we start with a *Variational Auto-Encoder* (VAE) (D. P. Kingma & Welling, 2013). We develop a performative tool that allows for the manipulation of hyperparameters in realtime while a Convolutional VAE trains on a live camera feed. In addition to gaining a number of insights with regards to how DNNs learn, and how the hyperparameters influence the results, the software that we developed also demonstrates great potential as a realtime visual performance tool.

After VAEs, in chapter 5: *Learning to see: Digital puppetry through realtime video transformation* we investigate *conditional Generative Adversarial Networks* (cGAN). Specifically, we build upon the application that we developed in the previous chapter, and using a custom pipeline based on *pix2Pix* (Isola et al., 2016), we develop a realtime video processing system that can transform a live camera feed with the aesthetic qualities of a large image based dataset. We demonstrate a number of different performative modes of interaction with this software. These include *live puppetry*, *augmented drawing*, and *realtime manipulation of parameters* using a graphical user interface or faders on a hardware interface.

In our final study, in chapter 6: *Deep Meditations: Latent storytelling*, we investigate *unconditional* Generative Adversarial Networks (GAN). Each of the four studies that we have presented so far, allow Meaningful Human Control, in a *Realtime Continuous* manner. However, in this study, we take a slight departure. We acknowledge the role that *non-realtime*, *non-continuous* modes of interaction have played in existing, traditional creative workflows. In particular, we look at the example of *video editing* as a form of *long-form narrative storytelling*. Taking a typical video editing workflow as a base, we present a method that allows for long-form story telling within a generative model's latent space, with meaningful control over the narrative

Finally, in chapter 7: *Conclusion* we summarise our findings, and speculate on potential future directions research in this field could take.

Chapter 2

Background

Our research is located at the intersection of a number of fields: **Deep Learning (DL)**; **Computational Art and Design (CAAD)**; and **Artistic, Expressive Human-Computer Interaction (AE-HCI)**.

As we mentioned in the previous chapter, we define **Creative DL** as the intersection of Deep Learning and Computational Art and Design — in other words, the application of DL techniques to the production of artistic works and creative media. We include both technical DL research within this field, and the artistic and/or design practices and research.

At the start of our research in 2014, there was not much activity within Creative DL. Today, this is a very active and rapidly growing area of research and practice. Within this area, our research focuses on introducing Artistic, Expressive Human-Computer Interaction; which we believe is still practically non-existent in this context. To be more precise, we investigate this through *Meaningful Human Control* and *Realtime Continuous Control for performative interaction*, which we discuss in section 1.3: *Meaningful Human Control* and section 1.4: *Visual instruments: Realtime Continuous Control* respectively.

In this chapter, we will provide the necessary history, terminology, definitions, and technical foundations that we will build upon in the remainder of this thesis.

We first discuss *classes of models*, namely **discriminative** and **generative** models. Since we spend a significant portion of this thesis working with generative models, we will spend some time laying the foundations of generative models in this chapter.

We follow this with a very brief history. We follow a number of independent threads which have separate origins, and eventually converge and merge. We consider 2015 to be the year that DL burst into the collective consciousness of artists, designers and creatives around the world — which led to the birth of **Creative DL**. This was initially fuelled by the viral online release of **Deepdream** (Mordvintsev et al., 2015) in June 2015, and its prevalent popularity on both social and mainstream media. This was shortly followed by the similarly viral online release and immense popularity of **Neural Style Transfer** (Gatys et al., 2015a) only a few months later, and **Deep Convolutional Generative Adversarial Networks (DCGAN)** (Radford et al., 2015), only a few months after that. Andrej Karpathy’s open-source character-level language

model **Char-RNN** (Karpathy, 2015a), based on (Graves, 2013), further cemented interest in DL within those with artistic and creative interests. This is not to say that no research was taking place within this field prior to 2015. However, 2015 was indeed the year that the relatively independent trajectories of DL and CAAD became much more intertwined.

We follow these threads, discussing both artistic and technical contributions. We do not attempt a thorough survey of DL. Instead we provide a very brief overview of key technical innovations, focusing on those events with significant cultural impact. In other words, these are **key milestones** within DL research that left the confines of academia and industry, and reached broader audiences to **affect the practices and discourse within the arts and creative sectors**. These include **Deepdream, Char-RNN, Neural Style Transfer, DCGAN, Pix2Pix, CyleGAN, AlphaGo, AlphaZero, Sketch-RNN, ProGAN, BigGAN, GANBreeder, ArtBreeder, StyleGAN and StyleGAN2**.

Finally, we present in more technical detail, an introduction to Deep Learning as will be necessary to parse this thesis. We start with the basics of Machine Learning, Artificial Neural Networks, loss functions, gradient descent and backpropagation, all the way through to modern deep architecture and methods that we use in this thesis, such as **Convolutional Neural Networks (CNN), Variational Auto-Encoders (VAE), Deep Convolutional Generative Adversarial Networks (DCGAN), Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs), and Monte Carlo Tree Search (MCTS)**.

2.1 Generative models

Before we go deeper into any of the topics, we will first define a few key concepts that will come up frequently in our discussion. We will regularly refer back to these ideas and build upon them in the following sections.

First, we discuss the two primary classes of Machine Learning models: *discriminative* vs *generative*. And then, since the majority of this thesis deals with generative models, we will examine them in more detail. We will leave implementation-level technical details for a later section, and discuss them in section 2.3: *Introduction to Deep Learning*. In this section, we will cover only *how* and *what* they can be used for.

Discriminative models learn to predict some output \mathbf{y} , given an input \mathbf{x} . More formally speaking, they model $P(\mathbf{y}|\mathbf{x})$. These can be used for **classification** purposes, where the model predicts a vector of probabilities over a number of class labels. In this case, the model only learns the boundaries between classes. Discriminative models can also perform **regression**, where the model predicts some real values. For example, what actions to take in a given situation, how much to steer an autonomous vehicle, what the price of a particular house might be given some conditions etc. A regression model might also predict the parameters of a parametric probabilistic model, for example a Gaussian Mixture Model (GMM) (Bishop, 1994).

In chapter 3: *Realtime sequence generation with continuous control*, we investigate how deep *discriminative* models can be used to *generate* images. In the chapters following that, we investigate a number of different *generative* deep models.

Generative models try to model observations by learning the joint distribution of the data. More formally speaking, they model $P(\mathbf{x}, \mathbf{y})$. In other words, generative models try to model the underlying dynamic processes that gives rise to the observations. When we take a sample from this distribution, we are effectively executing the function which models that dynamic process. Therefore, this generates new data that should be structurally and statistically similar to the observations. This also implies, that when we manipulate parameters of the generative model, we are manipulating parameters of the dynamic process. Therefore, this generates varying outputs that are consistent with the world from which the observations came from.

There are many different deep generative architectures, and in section 2.3: *Introduction to Deep Learning*, we will discuss a number of these in more technical detail. For now, we present some common characteristics.

2.1.1 Unconditional generative models

When we train a generative model on a dataset, for example consisting of many images, the model learns a function G that maps some vector \mathbf{z} to an output \mathbf{y} that resembles the training examples. In the architectures that we work with in this thesis, \mathbf{z} is typically sampled from a standard normal distribution $\mathcal{N}(0, 1)$. This can be expressed more formally as

$$G : \mathbf{z} \mapsto \mathbf{y} \quad \text{where } \mathbf{z} \sim \mathcal{N}(0, 1) \tag{2.1}$$

\mathbf{z} is a vector, with a dimensionality typically in the range 128–512 in the examples that we work with in this thesis. This is a compact, **latent representation** of the output \mathbf{y} , which might have a dimensionality many orders of magnitude higher. For example a 1024x1024 pixel colour image has a dimensionality of $1024 * 1024 * 3 = 3145728$. Yet it can be represented with a latent vector in 512 dimensions. This is effectively a compression ratio of 6144:1. For the generative model to learn such compact representations, the dataset must have significant amounts of structure and regularities. Furthermore, the model’s architecture must be designed specifically to maximise the ability of the network to identify and model these regularities and features. This is one of the major reasons as to why this is currently such an active area of research, and why there are so many different architectures available.

The generative model that we describe above in eqn. (2.1), in which the model’s output is not conditioned on any extra information, is an **unconditional generative model**. To generate an output, we simply sample a random $\mathbf{z} \sim \mathcal{N}(0, 1)$ and feed it through the model. In later sections, we will present a number of examples of unconditional generative models in subsection 2.3.17: *Variational Auto-Encoders (VAE)* and subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*.

2.1.2 Conditional generative models

It is also possible to condition the output of a generative model, on an additional input \mathbf{x} . This is known as a **conditional generative model**, and it can be expressed more formally as

$$G : \{\mathbf{z}, \mathbf{x}\} \mapsto \mathbf{y} \quad \text{where } \mathbf{z} \sim \mathcal{N}(0, 1) \quad (2.2)$$

It is quite common for the conditioning input \mathbf{x} to be a *class label*. For example, if \mathbf{z}_0 represents a specific location in latent space, $G(\mathbf{z}_0, \text{'dog'})$ would produce a specific image of a dog, that corresponds to that location \mathbf{z}_0 , while $G(\mathbf{z}_0, \text{'cat'})$ would produce an image of a cat, even though it is generating from the same latent representation. *BigGAN* (Brock et al., 2019) is an example of a conditional generative model, conditioned on class labels. Interestingly however, the authors note that *class leakage* may occur, whereby the visual characteristics of images from one class, might appear in the images generated when conditioned on another class, for example placing puppy eyes and nose on an image of a tennis ball.

Pix2pix (Isola et al., 2016), is a conditional generative model where the conditioning input \mathbf{x} is an image. This allows the model to generate new images, conditioned on an input image. We use this as a basis for the work that we present in chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, where we present an application that we developed which uses a conditional generative deep model to transform a live video feed in realtime.

2.1.3 Latent manipulations

One of the major motivations behind research into deep generative architectures, is to ensure that the latent representations learnt by the model, are well structured and disentangled (Bengio et al., 2013; Ridgeway, 2016). This ensures that given a latent representation \mathbf{z} where $G(\mathbf{z})$ produces an output \mathbf{y} , we can make certain meaningful manipulations on \mathbf{z} to produce a new latent representation \mathbf{z}' , such that $G(\mathbf{z}')$ produces \mathbf{y}' , which has some *meaningful manipulations* applied.

Latent representation recovery

Before we present examples of meaningful latent manipulations, we will first discuss the issue of *obtaining* latent representations to work with.

One of the simplest methods of obtaining latent representations, is to take **random samples** from the distribution. For the models that we work with, we typically *choose* the latent distribution during training, to be a standard normal distribution. For this reason, we can sample $\mathbf{z} \sim \mathcal{N}(0, 1)$. Once we have one or more latent representations obtained in this manner, we can perform the latent manipulations that we describe below. However, taking random samples in this manner, does not allow us to find the latent representation of a *given data point*. In other words, if for example, we are given an image, how can we find the latent representation of that image, such that we can apply meaningful latent manipulations to it?

There are a number of options with regards to addressing this.

Some generative models have **encoders** as part of their architecture. The encoder learns a function E , such that it can map an input \mathbf{x} to its latent representation \mathbf{z} . The transformation of the entire network can then be represented as $\mathbf{x} \mapsto \mathbf{z} \mapsto \mathbf{y}$. If the encoder and the generator are trained successfully, then the output of the Generator $\mathbf{y} \approx \mathbf{x}$, within the limitations of the generative capacity of the model, which in turn is dependent on many factors, such as the architecture, training hyperparameters, and size and diversity of the training data. A VAE is an example of a generative model with an encoder. We discuss VAEs in subsection 2.3.17: *Variational Auto-Encoders (VAE)*, and build upon them in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*. Flow-based generative models also have encoders (Dinh et al., 2016; D. P. Kingma & Dhariwal, 2018), and while they do allow for the latent manipulations that we describe below, they do not learn *compact* latent representations. In this respect, they are computationally significantly more expensive than VAEs, and so for our research, we decided to focus on the latter.

GANs, typically do not have encoders. However, depending on the complexity of the architecture and training data, **an additional encoder network can be trained**. An example of this is **VAE/GAN** (Larsen et al., 2016). While this approach showed great potential to begin with, VAE/GANs turned out to be notoriously difficult to train, and they quite often fail to converge. For this reason, there is generally not much research in this direction.

A more popular and successful approach to integrating some kind of encoder into GANs, is using optimisation techniques to **recover approximate latent representations** (Lipton & Tripathi, 2017; Karras et al., 2020). These methods are particularly effective for data points that are known to be within the distribution. For example, given a powerful image based generative model such as StyleGAN2 (Karras et al., 2020) trained on faces, it is possible to recover a latent representation for an image of a face which is not in the training data, but is cropped and aligned in a similar manner to images of faces which are in the training data. For data points that are outside the distribution, the ability of these methods to recover latent representations depends again on the generative capacity of the model.

Once we have used any of these methods to obtain one or more latent vectors, we can use meaningful latent manipulations such as those detailed below, to produce new latent representations that can be run through the generative model to produce meaningfully manipulated outputs.

Semantic latent vectors

It has been observed that there are often certain directions in the latent space of a generative model, such that moving a latent representation in that direction, applies meaningful manipulations to the output data point. For example in Radford et al. (2015), the authors train a DCGAN on a large dataset of faces. They select a number of latent representations for images of faces wearing glasses, and they calculate the mean to obtain an average face-*with*-glasses latent vector $\mathbf{z}_{\text{avg-face-with-glasses}}$. They do the same to obtain an average face-*without*-glasses latent vector $\mathbf{z}_{\text{avg-face-without-glasses}}$. Subtracting one from the other produces a **semantic vector** $\mathbf{z}_{\text{add-glasses}} = \mathbf{z}_{\text{avg-face-with-glasses}} - \mathbf{z}_{\text{avg-face-without-glasses}}$ which indicates

the direction of *adding glasses to a face*, in the latent space. Adding this semantic vector $\mathbf{z}_{\text{add-glasses}}$ to the latent representation of any new image of a face, produces a latent representation $\mathbf{z}_{\text{new-face-with-glasses}} = \mathbf{z}_{\text{new-face-without-glasses}} + \mathbf{z}_{\text{add-glasses}}$ such that $G(\mathbf{z}_{\text{new-face-with-glasses}})$ produces an image of that face wearing glasses, even if that image does not exist in the training set. The authors demonstrate this with other examples including smiling and rotating.

Using semantic vectors to meaningfully manipulate images is an area that shows great potential and is gaining popularity in recent years. The artist and educator Tom White was one of the first researchers to explore this artistically in his *Smile Vector* Twitter-bot (White, 2016b). In this project, White trains a VAE on images of faces, and identifies semantic latent vectors such as *smiling*. The public can interact with the Twitter-bot, to submit their own images. The bot adds and subtracts the *Smile Vector* to the user-submitted images, to make the faces smile more or less. As generative models, and DCGANs in particular, are starting to produce more realistic outputs, research in semantic latent vector *discovery* is growing (Simon, 2019; Abdal et al., 2019; Karras et al., 2019, 2020; Härkönen et al., 2020).

Latent interpolation

An important and related concept to semantic latent vectors, are **latent interpolations**. Interpolating between different latent representations, produces outputs which transform gradually and in some meaningful manner. For example, the intermediate latent representations z_i that we obtain, when slowly interpolating from a latent representation \mathbf{z}_l of a face looking to the left, to the latent representation \mathbf{z}_r of a face looking to the right, will produce images of the face gradually turning from left to right. Likewise, slowly moving between the latent representations of a face frowning, to the same face smiling, effectively generates an animation of the face starting with a frown, and gradually smiling more and more. When we interpolate between very seemingly unrelated latent representations, the model produces outputs in accordance with the set of rules that it learnt, which may or may not make sense to us. For example, interpolating between the latent representation of a tree and a chair, will result in some kind of *morphing* images, which are difficult for us to predict, and depends entirely on the structure of the latent space. However, modern deep generative models are able to learn very well structured latent representations, such that these interpolations do produce remarkable results. In the case of image based generative models, fully leveraging the visual qualities of the images. We present some examples in chapter 6: *Deep Meditations: Latent storytelling*.

Latent walks

Videos of latent space interpolations have become very popular, as they can demonstrate the capabilities of the underlying generative model quite well. As a result, it is not uncommon for researchers who investigate novel generative architectures, to release latent space interpolation videos along with their research¹.

Typically, these interpolation videos are created without much meaningful human intervention. This is to say, they tend to be either very short clips, simply morphing between two states

¹For example, an accompanying latent space interpolation video for StyleGAN2 (Karras et al., 2020) <https://www.youtube.com/watch?v=c-NJtV9Jvp0>

such as smiling or not smiling. Or the longer clips tend to be entirely *random walks*. In other words, a number of random latent representations are sampled from the latent distribution, and then they are interpolated between. This is primarily because there are no tools currently available, which allow for the design of trajectories in the high-dimensional space of latent representations. This is a topic that we address in our work. Namely, we propose a method, and develop a tool, that allows users to construct very specific trajectories, such that *users can tell stories in latent space with Meaningful Human Control over the narrative*. We discuss this with more detail in chapter 6: *Deep Meditations: Latent storytelling*.

It's worth mentioning, that these latent manipulation methods that we describe above, are applicable to not only images, but they can be applied to potentially any domain. In *GANSynth* (Engel et al., 2019), the authors use latent manipulations on short clips of raw audio, to interpolate between the timbre and pitch of musical instruments. *Skip-thought vectors* (Kiros et al., 2015) are latent representations of entire sentences, and in the open-source demo *neural-storyteller* (Kiros, 2015), the author applies semantic latent manipulations to ordinary text, to manipulate them into the style of romance novels. For the purposes of our work that we present in this thesis however, we are currently focusing on images.

2.1.4 ‘Generative’ terminology in different domains

It is important at this point, to take a slight digression, and discuss this terminology, especially in a cross-disciplinary context such as our research. The term *generative* is used frequently in this thesis. However, it refers to slightly different concepts in different domains, which are relevant to this thesis:

ML, inherits much of its terminology and concepts from *statistics*. This is also where the definition of *generative model*, as we described above originates.

In section 2.2: *Very brief histories*, we discuss the notion of **generative art**, and the broader notion of **generative media**, which includes art, music, design, architecture, poetry, text and any other creative medium. In the computer graphics literature, this is often referred to as **procedural**. In the art world, it is primarily known as *generative*, but can also be referred to as **algorithmic art**. Generative art or media, and generative models refer to very different concepts, which we will unpack shortly. However, before we do so, it is important to also mention another similar sounding, but conceptually distinct term.

In engineering, product design and architecture, **generative design**, refers to a process of iterative, parametric design involving constraints and goals. It is a family of methods used primarily for solving design and engineering problems. The designer will set constraints relating to materials, space, weight, volume, dimensions, and costs, along with functional and/or aesthetic objectives. Through an iterative process, the generative design system tries to optimize for a number of different solutions.

As we can see, these three definitions refer to quite different, but related concepts. Most notably, both *generative models* and *generative design*, can be included within *generative media*. In other words, an image sampled from a generative Machine Learning model, or produced through a generative design process, could be considered to be *generative media* or *generative*

art. However, the reverse is not always true. For example, an image produced via particle systems, circle packing, and Delaunay triangulation, is considered generative media. But it is neither generative design, nor a generative model.

It's also important to bear in mind, that not *all* Machine Learning systems that produce generative media, are generative models. For example, a DCGAN that we will present shortly, *is* a generative model. Taking a random sample from the latent distribution, a multivariate Gaussian, produces an image. This is generative media, sampled from a generative model. However, Deepdream, as we will shortly discuss, is *not* a generative model. It is an algorithm in which we use a method known as *activation maximization*, on a *discriminative* model designed and trained for *classification*. We can generate images using Deepdream, so the system *is* generative, from a *generative media* point of view. However it is not a *generative model*, from a ML or statistics point of view.

These distinctions are subtle, but they become important when communicating with researchers and practitioners from different fields.

In this thesis, we do not touch upon *generative design*. This is because we are more interested in realtime, expressive interaction for artistic expression; while generative design is generally more focused on solving design problems with engineering constraints. However, both *generative media* and *generative models* are integral to our research. In this text, whenever we use the term *generative model*, we are explicitly referring to exactly that, a generative statistical model (or more specifically in our case, a generative Deep Neural Network). Whenever we use the term *generative system*, we are referring to a computational system which is capable of algorithmically generating media. This system may be a generative model, such as a DCGAN. Or it may be a system comprising of multiple components, one of which may be a non-generative Deep Neural Network, such as in the case of Deepdream.

2.2 Very brief histories

In this section, we discuss the separate histories of a number of different threads that relate to **Deep Learning (DL)**, **Computational Art and Design (CAAD)**, and **Artistic, Expressive Human-Computer Interaction (AE-HCI)**. We initially trace these threads independently, and we follow them through where we observe points of convergence.

Within the greater context of art history, **computational art** makes a very recent appearance, spanning only a single human lifetime. Despite this, the genre has grown rapidly, and has expanded into many overlapping sub-genres. Just to name a few, these include generative art, new media art, interactive art, net art, post-internet art, digital art, software art, algorithmic art, data art, evolutionary art, robotic art, techno art, glitch art, video game art, bot art, AI art, and many more. In this section, we only touch upon aspects of computational art that connect themes which are directly relevant to our research, such as *expressive interaction*, *Artificial Intelligence* and *Machine Learning*. For a more comprehensive review of computational art in general, please refer to (Levin, 2000; Grierson, 2005; Bailey, 2020).

2.2.1 Generative art

The term **generative art** (and the broader term **generative media** — which includes art, music, design, architecture, poetry, text and any other creative medium), refers to works created as a result of working with rule-based and semi-or-fully autonomous systems. Instead of designing and producing the final artefact directly, the generative artist designs a *process*, which in turn produces the final artefact. The lineage of generative art exists outside of purely computational practices, and can be traced to composers such as Iannis Xenakis, Steve Reich, John Cage, Terry Riley, Brian Eno; and artists such as Lillian Schwartz, Sol Lewitt, Sonia Sheridan; Muriel Cooper and Nam June Paik. Generative art also includes artists working with biological systems, such as Tomás Saraceno’s works with spiders and spider webs, and Heather Barnett’s work with slime mould. It is worth noting, that the term *generative art*, is very different to the terms **generative models** — which we use frequently in this thesis — and **generative design**. To avoid confusion, we expand on this in subsection 2.1.4: *‘Generative’ terminology in different domains*.

As we mentioned in chapter 1: *Introduction & Motivations*, even before the concept of generative art existed, Lady Ada Lovelace had already foreseen in 1843, the impact that computers would have, in the creation of ‘algebraical patterns’ and ‘elaborate and scientific pieces of music’. Indeed, the introduction of computers — first analog, and then digital — radically expanded the field of generative art, and led to the birth of many new genres, collectively known under the umbrella title of **computational art**.

The use of computers for the purposes of making art dates back to the 1950s and 1960s. John Whitney built DIY analog computers from World War II M5 and M7 targeting computers, and anti-aircraft machinery (Alves, 2005). With these DIY computers, Whitney created abstract, animated, visual works, which were not only pioneering technically, leading to the birth of computer graphics and special effects in scenes such as Stanley Kubrick’s 2001: A Space Odyssey, but in addition they pioneered the field of computer-aided audio-visual composition (Grierson, 2005). His work continues in the tradition of experimental abstract animators and filmmakers such as Normal McLaren and Oskar Fischinger. Throughout the 1960s and 1970s, he was joined by software artists such as Paul Brown, Vera Molnar, Manfred Mohr, Lillian Schwartz, Frieder Nake, Grace Hertlein, Larry Cuba, George Nees and many more, collectively giving birth to the *digital* generative art movement.

2.2.2 AI and ML in art, pre-Deepdream

It was Harold Cohen’s AARON software from 1973 (Cohen, 1973) which first explicitly brought *Artificial Intelligence* into the world of computer art. In this respect, it can be considered the first example of **AI art**. AARON was ostensibly a piece of software, written to understand colour and form. Cohen often spoke about the concept of *training* his software (Cohen, 1994). However, he uses the term rhetorically. The *learning* that takes place within AARON, is not what we consider to be true *Machine Learning* as we will discuss in the following sections. AARON does not *learn through experience*. AARON does not learn through observations, or analysing data. Instead, when Cohen wanted AARON to learn something new, and perform a

particular new task, Cohen had to first analyse the task himself. Cohen identified the sets of rules that governed the behaviour that he wanted AARON to perform. And then he explicitly programmed those rules into AARON, so that AARON could perform the desired behaviour. Often, these were very complex sets of rules, that took years for Cohen himself to decipher and learn, before he could program them into AARON (Cohen, 2006).

As computers started becoming more accessible in the 1980s and 1990s, this brought a new generation of computer graphics artists working with Artificial Intelligence. These include William Latham and Stephen Todd (Todd & Latham, 1992), Karl Sims (Sims, 1994) and Scott Draves (Draves, 2005). Inspired by Darwinian evolution, these artists appropriated **genetic algorithms**, to pioneer the discipline of **evolutionary art**. Genetic algorithms do *learn from experience*, and are technically considered to be Machine Learning algorithms, and for this reason, these works from the 1980s and 1990s can be thought of as among the earliest examples of *true Machine Learning based artworks*.

During this same period, David Cope developed an algorithm for composing music. His *Experiments in Musical Intelligence* (EMI) began in 1981, and he developed it for many decades, until he eventually patented the algorithm *Recombinant music composition algorithm* (Cope, 2010). Using this algorithm, Cope generated musical sequences in the style of many classical composers, such as Bach, Vivaldi, Beethoven, Mozart, Chopin and Debussy. His latest software *Emily Howell* has albums being released under its name. The algorithm he uses is based on *Markov chains*, an early 20th century probabilistic model capable of capturing temporal patterns. We will leave the technical discussion for later sections,

Throughout the late 1990s and early 2000s, François Pachet, based at the Sony Computer Science Laboratory in Paris, developed the *Continuator* (Pachet, 2003). Also based on Markov models, this is an interactive musical system that is capable of *learning in realtime*, stylistic characteristics of music, such as rhythm, beat, and harmony. The system can improvise and accompany a live musician, or given a sequence of music, it can *continue* in a similar style.

2.2.3 Interactive media art

The artists mentioned above, were primarily concerned with the algorithmic creation of images and/or sound, and can be thought of as practicing in the lineage of generative art.

Since the 1960s, Myron Kruger was primarily interested in *gesturally interactive* computer artworks set in **Responsive Environments**. Kruger developed his large-scale immersive, responsive installations throughout the 1960s, 1970s and 1980s, culminating in his seminal artificial reality environment *Videoplace* (Krueger et al., 1985). Videoplace tracked users with cameras, enabling them to interact with virtual objects in the scene using projectors.

David Rokeby's *Very Nervous System*, first developed in 1986, explores similar themes of gestural full body interaction, in this case to generate music (Rokeby, 1986). Throughout the 1980s and 1990s, artists such as Ed Tannenbaum, Scott Snibbe, Michael Naimark, Golan Levin and Camille Utterback created **interactive media artworks**.

While many of these works might not be immediately considered *AI based* works, they use sensors, cameras, projections and robotics, to create immersive and interactive environments that *respond intelligently to a users actions*. They often incorporated complex algorithms, both

on the *input and sensing* side, for example using computer vision or voice analysis techniques; and on the *output* side, for example mapping inputs to complex behaviours of images, sounds, or motors. And using these complex algorithms, they provide *gestural and expressive interaction for the realtime creation and manipulation of images and sounds*.

2.2.4 Visual instruments

We have already discussed Visual Instruments in section 1.4: *Visual instruments: Realtime Continuous Control*. Similar to Louis-Bertrand Castel's *Ocular harpsichord*, many mechanical visual instruments were built throughout the 19th and early 20th century. These include Frederic Kastner's *Pyrophone* in 1869, Bainbridge Bishop's high voltage powered light emitting pipe organ in 1877, Thomas Wilfred's *Clavilux* in 1919, George Hall's *Musichrome* and Charles Dockum's *MobilColor Projectors* in the 1930s, and Oskar Fischinger's *Lumigraph* in the 1940s. For a more in-depth discussion regarding these, please refer to (Levin, 2000). For a more in-depth discussion regarding video synthesizers such as the *Bill-Etra* or *Paik-Abe*, please refer to (Collopy, 2014).

2.2.5 Convergence

From the late 1990s onwards, we see an intersection in these initially disparate fields of computational art.

With the birth of the web, and the introduction of first Macromedia Director, and then Macromedia (Adobe) Flash, a new generation of digital generative artists blossomed. These include artists such as Lia, Joshua Davis, Jared Tarbell, Mario Klingemann, and many more.

Around the same time, John Maeda and his *Aesthetics and Computation* research group at MIT Media Lab developed the *Design by Numbers* program aimed at teaching programming to artists. Developing this further, and inspired by Muriel Cooper's *Visual Language Workshop*, Casey Reas and Ben Fry's open-source creative programming environment *Processing* was born in 2001.

With the introduction of these new artist-friendly programming environments, and perhaps more importantly — *their respective communities*, the generative and new media art movements converged and grew exponentially.

Initially, the majority of these artists and works were situated more in the lineage of *generative art*. However, as more open-source tools and communities became available over the years, different technologies such as computer vision and physical computing became more accessible to a wider and more diverse audience. This encouraged the rapid growth of *interactive media art* in the early-mid 2000s. Some of these tools and communities include openFrameworks, Cinder, vvvv, Max/MSP/Jitter, PureData, SuperCollider, TouchDesigner, QuartzComposer, Three.js, p5.js and many more.

2.2.6 Creative DL \leftarrow AI Art \cup Creative AI

With the emergence of *Deepdream*, *Neural Style Transfer*, *DCGANs* and *Char-RNN* in 2015, Deep Learning became a dominant paradigm within CAAD, and **Creative DL** was born.

Over the next five years, alongside the technical innovations being developed by DL researchers; artists, designers and creative developers also became very interested in the field. The terms **AI Art** and **Creative AI** began to emerge, referring to both the methods, and the community of people who were interested in DL for artistic or other creative purposes. One could point out that there are subtle differences between *AI Art*, which is focused on more purely aesthetic applications, and *Creative AI*, which is broader and considers *any* creative applications including design and architecture.

Despite having ‘AI’ in their names, these terms don’t necessarily refer to ‘Artificial Intelligence’ in the general sense, but are typically more focused on specific modern DL technologies. For this reason, in this thesis we group them under the single title **Creative DL**.

In this respect, the *Creative DL* community is very young. However, after a very rapid growth, today it is thriving. This rapid growth is due to a number of reasons.

The continual release of new technological innovations, and the accompanying viral social media response is most probably a strong driving force. We will summarise these in the next section. Another factor for the rapid growth is an increase in accessibility. Some of the key developments include the highly popular website *Artbreeder* (Simon, 2019), which we will discuss shortly; the *Machine Learning For Artists* (ML4A) initiative led by Gene Kogan ², a collection of free educational resources devoted to familiarising artists with state-of-the-art deep DL; and DL based artist-in-residency programs such as Google’s *Artists & Machine Intelligence* (AMI) and *Arts & Culture* residency. Finally, efforts from community leaders such as Luba Elliot and AIArtists.org³ helped organise and bring the community together and spread the word.

While the field is incredibly popular today, some of the earliest adopters of Creative DL (from a purely artistic perspective, we will cover more technical/creative developments shortly) include Addie Wagenknecht, Alex Champanard, Alex Mordvintsev, Alexander Reben, Allison Parrish, Anna Ridler, Gene Kogan, Georgia Ward Dyer. Golan Levin, Hannah Davis, Helena Sarin, Jake Elwes, Jenna Sutela, Jennifer Walshe, Joel Simon, JT Nimoy, Kyle McDonald, Lauren McCarthy, Luba Elliott, Mario Klingemann, Mike Tyka, Mimi Onuoha, Parag Mital, Pindar Van Arman, Refik Anadol, Robbie Barrat, Ross Goodwin, Sam Lavigne, Samim Winiger, Scott Eaton, Sofia Crespo, Sougwen Chung, Stephanie Dinkins, Tega Brain, Terence Broad and Tom White.

It is within the intersection of all of these threads that we situate our research, where we aim to bring more *Meaningful Human Control*, and *Realtime Continuous Control* to the methods available to artists, designers and creatives working within this area.

2.2.7 Creative Deep Learning — from a cultural perspective

The history of Deep Learning spans many fields, including not only Machine Learning and Artificial Intelligence; but also information theory, computational complexity theory, control theory, neuroscience, statistics, calculus, philosophy, psychology, cognitive science, and the social sciences.

²<https://ml4a.github.io>

³<https://aiartists.org>

For this reason, we will not attempt to present a historic survey of Deep Learning, and instead we will provide a very brief overview of the key technical innovations, from the point of view of *artistic and cultural significance*. In other words, below we identify a number of significant events and milestones within DL research, that left the confines of academia and industry, reached broader audiences through mainstream and social media, and affected the discourse within the arts and creative sectors. We focus primarily on *visual* outputs, as that is the focus of our thesis. This topic could become a thesis in itself, so we will keep it very brief and only provide the top-level headlines.

We will not discuss technical contributions or details in this summary, and we leave that for section 2.3: *Introduction to Deep Learning*. For a thorough historic survey of the topic, please see Schmidhuber (2015), and for more technical details, please see I. Goodfellow et al. (2016).

Deepdream (2015)

As we previously mentioned, the first significant milestone which brought DL into the imaginations of the mainstream creative world, came in 2015 with what was initially known as **Inceptionism**, and later rebranded as **Deepdream** (Mordvintsev et al., 2015).

Deepdream sits in a lineage of research that is centred around the quest to understand how deep image classification models work, by *visualising their hidden layers* or *inverting them* (Erhan et al., 2009; Simonyan et al., 2013; Zeiler & Fergus, 2013; Mahendran & Vedaldi, 2014; Dosovitskiy & Brox, 2015; Nguyen et al., 2015; Olah et al., 2017; Mordvintsev et al., 2018; Olah et al., 2018).

The Deepdream algorithm, uses a *gradient ascent optimisation method*⁴, similar to those we reference above, to generate images that maximise activity on particular hidden neurons or layers. The output generated by this process, is fed back into the system to create feedback loops that incrementally amplify the activity. Combining this with image transformations such as scale or translation, at every iteration, allows for the creation of endless fractal-like animations and ‘hallucinations’ of abstract, but recognisable, imagery.

⁴We discuss this in more technical detail in subsection 2.3.10: *Gradient descent and backpropagation*.



Figure 2.1: Still image from video “*Deepdream is blowing my mind*” (2015) by Memo Akten.

Creative Deep Learning as viral marketing campaign

Deepdream became incredibly viral and popular on social and mainstream media. For a brief moment during the summer of 2015, certain corners of the internet were flooded with the *puppy-slugs* and *bird-lizards* of Deepdream⁵, to which our own experiments did contribute⁶. The first version of the code that was released was notoriously painful to setup due to a complex network of dependencies (colloquially known as *dependency hell*). However, shortly after, a slew of easy to use mobile apps, websites and business services were launched to ‘Deepdream-ify’ user submitted images⁷⁸⁹¹⁰ with the hopes of cashing in on this new ‘AI powered art craze’.

Google themselves recognised the enormous PR and marketing potential of Deepdream, and quickly launched an exhibition in San Francisco, collaborating with the well-established *Gray Area Foundation For The Arts*¹¹, a non-profit organisation with a mission to “*apply art and technology to create social and civic impact through education, incubation and public events*”.

This exhibition, called “*DeepDream: The art of Neural Networks*”, launched in February 2016. In fact, our own work “*All watched over by machines of loving grace: Deepdream edition*” (2015)¹² was included in this exhibition, and was the highest selling work at the auction, for \$8000 (Fig. 2.2).

⁵<https://twitter.com/search?q=deepdream>.

⁶<https://twitter.com/search?q=from%3Amemotv+deepdream>

⁷<https://dreamscopeapp.com/deep-dream-generator>

⁸<https://deepdreamgenerator.com>

⁹<https://apkpure.com/deep-dream/com.deepdreamnow.app.deepdream>

¹⁰<https://www.ostagram.me>

¹¹<https://grayarea.org/event/deepdream-the-art-of-neural-networks>

¹²<http://www.memo.tv/works/all-watched-over-by-machines-of-loving-grace-deepdream-edition>



Figure 2.2: “All watched over by machines of loving grace: Deepdream edition” (2015) at the “Deep-Dream: The art of Neural Networks” exhibition at the Gray Area Foundation, February 2016.

The exhibition further fuelled mainstream interest in Deep Learning, and gained significant attention from mainstream publications such as Gizmodo¹³, Vox¹⁴, Wired¹⁵, The Wall Street Journal¹⁶, The Washington Post¹⁷, The Verge¹⁸, and The Guardian¹⁹.

In fact, as we were involved in this exhibition, we are very aware of the intense amount of human labour and creative decisions involved in the production of these works. In that respect, our research in this thesis is in some way a reaction to this press, and specifically to the narrative that was constructed in the PR campaign promoted by Google.

As can be seen by looking at the headlines in Fig. 2.3, Google’s PR campaign had clearly decided to spin the story such that “Google’s AI is creating art”. This was further exacerbated by the fact that the stories themselves were mostly focused on Google’s AI technologies, with very little mention of the artists, nor the exhibition’s collaborating partner, The Gray Area Foundation For The Arts.

This would start a trend that would follow in subsequent years: using DNNs in the production of creative media such as images or sounds as material for online viral marketing. Nvidia’s ProGAN (Karras et al., 2017), StyleGAN (Karras et al., 2019) and StyleGAN2 (Karras et al., 2020) are great examples of this. These are state-of-the-art generative image models that are currently able to produce the most cutting-edge and photo-realistic results within the field. The authors freely share their source code. However, needless to say, they suggest that in order to achieve the results that they demonstrate in their papers and videos, they recommend using Nvidia’s DGX-1 with 8x Tesla V100 GPUs — Nvidia’s own PC designed for Deep Learning, costing in the order of hundreds of thousands of dollars. Arguably, the cost of a small Deep Learning research team, combined with state-of-the-art results in image generation, and a hype-driven online community hungry for dazzling content, can be a very cost-effective form of PR and marketing.

Having said that, the research behind Deepdream itself is something that we do find fascinating, from multiple perspectives. This includes the point of view of trying to understand *how* and *what* DNNs learn from a technical perspective. This line of inquiry has led to equally fascinating research, often involving the same group of researchers (Olah et al., 2017; Mordvintsev et al., 2018; Olah et al., 2018). And we also find the research behind Deepdream fascinating from the point of view of using machines that learn, to reflect on how humans make sense of the world. A video and article that we wrote on this topic called “*Deepdream is blowing my mind*” (2015)²⁰ was shared widely on the internet, and we touch upon this in more detail throughout this thesis.

However, while we have experimented significantly with Deepdream, generating images in

¹³<https://gizmodo.com/this-google-dream-bot-inspired-artwork-is-mind-blowing-1761049728>

¹⁴<https://www.vox.com/2016/2/27/11588302/googles-tripply-ai-neural-nets-put-on-an-art-show>

¹⁵<https://www.wired.com/2016/02/googles-artificial-intelligence-gets-first-art-show>

¹⁶<http://blogs.wsj.com/digits/2016/02/29/googles-computers-paint-like-van-gogh-and-the-art-sells-for-thousands>

¹⁷<https://www.washingtonpost.com/news/innovations/wp/2016/03/10/googles-psychedelic-paint-brush-raises-the-oldest-question-in-art/?postshare=921457638736588>


¹⁸<https://www.theverge.com/google/2016/3/1/11140374/google-neural-networks-deepdream-art-exhibition-san-francisco>

¹⁹<https://www.theguardian.com/artanddesign/2016/mar/28/google-deep-dream-art>

²⁰<http://www.memo.tv/works/deepdream-is-blowing-my-mind>

GIZMODO

This Google Dream Bot-Inspired Artwork Is Mind Blowing



Courtesy of Memo Akten / Gray Area Foundation

Last summer, the internet was overrun with six-eyed dog faces, human legs that are actually slugs, and other images reminiscent of the day you ate magic mushrooms and feverishly explored your kitchen floor. In fact, these were the dreams of an AI developed by Google. And it was only a matter of time before the technology inspired new forms of art.

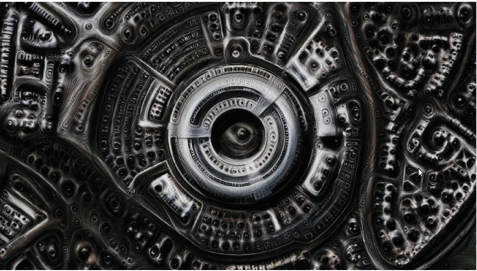
That day, my friends, has come. The Gray Area Foundation, in collaboration with Google Research, has put together the first of several art exhibits that make use of biologically-inspired forms of cognitive called artificial

The Washington Post

Innovations

Google's psychedelic 'paint brush' raises the oldest question in art

By Matt McFarland March 10



(Courtesy Memo Akten)

A recent San Francisco art show hosted by Google included the typical trappings — a big crowd, striking pieces of art and expensive price tags. But there was one noteworthy distinction of this particular art show — all of the pieces were created with the help of Google's computer science

Most Read

- 1 After Baton Rouge, a weary fear builds among those who protect and serve
- 2 Three police officers killed, three others wounded in Baton Rouge shooting, attacker identified
- 3 Sister of slain Baton Rouge officer Montrell Jackson: 'It's coming to the point where no lives matter'
- 4 Police union calls for open carry gun ban as fears of violence mount ahead of GOP convention
- 5 Aren't more white people than black people killed by police? Yes, but no.

Our Online Games

Play right from this page


Matchings Sunday Crossword by

Google's Artificial Brain Is Pumping Out Trippy—And Pricey—Art

DESIGN DEAR SCIENCE SECURITY TRANSPORTATION

BASEBALL BUSINESS 03/28/16 7:00 AM

GOOGLE'S ARTIFICIAL BRAIN IS PUMPING OUT TRIPPY—AND PRICEY—ART



17/14 GCMH MEMO AKTEN

ONE.A NEW COOP COMPANY

10/27/15 0.00% Hang Seng + 2734.08 0.55% U.S. 10 Yr 0/32 Yield 1.54% Crude Oil + 43.91 -0.05% Yen + 105.58 0.64% DAX + 1876.55 0.05%


THE WALL STREET JOURNAL

Home World U.S. Politics Economy Business Tech Markets Opinion Arts Life Real Estate

DIGITS

Google's Computers Paint Like Van Gogh, and the Art Sells for Thousands

By GEORGIA WELLS Feb 25, 2016 7:39 pm ET



Google engineers applied an algorithm to the artistic process, giving art quantitative and mathematical attributes. Computers, with human helpers, used a technology called neural networks to make 20 works of art for the show. [SEE OR SHARE CONTENT](#)

Google is trying to make art freaks out of computer geeks.

Recommended Videos

- 1 Garage Apartments Get an Upgrade
- 2 The Summer Deck Has Its Moment in the Sun
- 3 A Top Chef's Summer Entertaining Tips
- 4 Jesse Jackson Reacts to Baton Rouge Shooting
- 5 The Technology Behind 'Pokemon Go'

Most Popular Articles

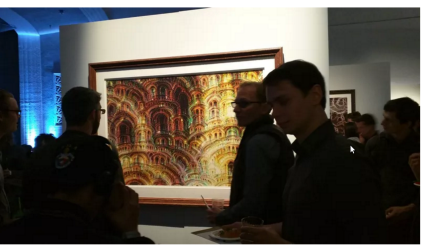
- 1 Three Officers Dead in Baton Rouge Shooting

recode TRENDING TOPICS WRITERS PODCASTS EVENTS

Google's Trippy AI Neural Nets Put On an Art Show

Can a machine outdo Picasso?

BY MARK BERGEN @MARKBERGEN FEB 23, 2016, 1:00 PM



Mark Bergen / Recode

Almost every day, machines outmatch humans on some task. They identify faces and places better than us. They beat us at beveling board games.

the guardian

UK world sport football opinion culture business lifestyle fashion environment tech travel

Art

Can Google's Deep Dream become an art machine?

The company's neural network has created a slew of beautiful and at times terrifying images, and is being harnessed to create unique artwork.

Alex Rayner

Monday 28 March 2016 22:51 BST



Monograph (aperture) art created by Deep Dream. Photograph: Deep Dream

Over the last weekend in February one of Google's computer science departments, Research at Google, co-hosted Deep Dream: the art of neural networks, with the Gray Area Foundation, a San Francisco not-for-profit organisation that fosters collaborations between the arts and technology.

Most popular

- 1 Baton Rouge man suspected of killing three police officers 'was former marine' - live...
- 2 Gavin Long, named as Baton Rouge gunman, was marine with online alias
- 3 Six wealthiest countries host less than 9% of world's refugees
- 4 Barack Obama condemns killing of three police officers in Baton Rouge shooting
- 5 New attacker's movements studied as...

The Guardian's voice is needed now more than ever. Support our journalism for just \$49 (£49) per year.

Figure 2.3: A selection of press covering the “DeepDream: The art of Neural Networks” exhibition at the Gray Area Foundation, February 2016.

this manner is a painfully slow process, as it requires an optimisation consisting of many hundreds or even thousands of iterations just to produce a single image. While there is scope for *Meaningful Human Control*, it is very far from *realtime*, or *continuous* control. Furthermore, as a reaction to the press coverage of the “DeepDream: The art of Neural Networks” exhibition, we choose to focus our research specifically on the significance of the *role of the human*, in a collaborative relationship with a generative system powered by a Deep Neural Network. This is very much in contrast to the narrative put forward by Google’s PR campaign, and absorbed by the mainstream media, that “Google’s AI is making art”.

It is from this perspective, that *Meaningful Human Control*, and *Realtime Continuous Control* emerge as points of emphasis in this thesis.

Char-RNN

Another significant milestone came in 2015, in a different domain. While the underlying research is from 2013 (Graves, 2013), it was in 2015 that Andrej Karpathy published a very influential blog post with the title “*The Unreasonable Effectiveness of Recurrent Neural Networks*” (Karpathy, 2015a), and released an open-source LSTM RNN implementation for training character-level language models, which he called **Char-RNN**. This software was later ported to different frameworks, and has been used by countless people to generate text in the style of Shakespeare, cooking recipes, rap lyrics, Obama speeches, the bible, Seinfeld episodes, and many more (Karpathy, 2015b). This software was also used by researchers such as Sturm (2015) to generate midi notes in the style of folk music, and is used extensively by artists such as Ross Goodwin²¹ and Allison Parish²² to produce very creative, generative text works. In section 3.3: *Realtime interactive text generation with RNN ensembles*, we go back to the original research that Char-RNN was implemented from (Graves, 2013), and we develop a new version that interactively allows Realtime Continuous Control over the style of the text generated.

Neural Style Transfer (2015)

Yet another significant milestone came in 2015, in the form of **Neural Style Transfer**, which became popular on social media with the hashtag **#StyleNet**²³

The basic idea behind **Style Transfer**, is to apply the *style* of one image, onto another image which provides the *content*. A typical example might be to apply the style from a Monet painting, to a photograph of a portrait or a landscape. In this case, the photograph will be transformed to look as if it were painted by Monet. While computational approaches to such artistic stylisations have a rich history (Hertzmann, 2001; Kyprianidis et al., 2013), a data-driven approach using DNNs were first demonstrated in (Gatys et al., 2015a). This original method of **Neural Style Transfer**, used an optimisation approach to generate an image, such that it satisfied both constraints of having *stylistic information* from the *style image*, and *content information* from the *content image*. In this respect, similar to Deepdream, generating images is again very slow, and generally takes in order of minutes for a single image. Furthermore, this

²¹<https://rossgoodwin.com/>

²²<https://www.decontextualize.com/>

²³<https://twitter.com/search?q=stylenet>

method deals with a *single* style image. In other words, a photograph, can be stylised with only one style image.

In Deepdream, the majority of images generated share a very strong common aesthetic, namely puppy-slugs and bird-lizards. For this reason, while it became very popular incredibly quickly, its popularity — at least within the broader Creative DL community — faded equally quickly. However, Neural Style Transfer, allows for much easier customisation of the aesthetics, since the *aesthetics* is determined by the *style image*. In other words, simply selecting different style images, allows a user to radically alter the look of the generated images. For this reason, the popularity of Style Transfer did not diminish as quickly as Deepdream, and artists such as Sofia Crespo²⁴ and Chris Rodley²⁵ have developed unique techniques integrating Style Transfer into their practice to make works that carry their personal expressive signature.

Many improvements were developed such as fast style transfer (Johnson et al., 2016), multiple styles (Cui et al., 2017), texture synthesis (Risser et al., 2017), applications to videos (Ruder et al., 2016) and spherical images (Ruder et al., 2018). **Image analogies**, as first put forward by (Hertzmann, 2001), also became a popular area of research. This involves using low-colour *semantic maps*, to generate high-fidelity images such as photographs or paintings. For this reason, within the Style Transfer discourse, this became known as **Semantic Style Transfer** (Champandard, 2016).

This topic would later be taken over by a very different technical approach using GANs, which we will discuss shortly. And it is this implementation with GANs, that we build upon for the work that we present in chapter 5: *Learning to see: Digital puppetry through realtime video transformation*. Style Transfer has been adapted to run in realtime (Johnson et al., 2016), and versions of it can support multiple styles (Cui et al., 2017). However, they are still limited to the order of *tens* of different style images, while we are interested in the question regarding what can deep generative models learn across a dataset of *tens of thousands* of images. Furthermore, even if the number of style images were not an issue, typically within Neural Style Transfer, a user explicitly selects one or more images from which the style is extracted and used. In other words, the style of each image is modelled individually. However, in chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we are again interested in modelling the aesthetic qualities across the entire dataset of tens of thousands of images.

It is also worth noting, that the research being conducted with the field of Style Transfer, also served as inspiration for similar research in different domains. These include Neural Style Transfer for Audio Spectrograms (Verma & Smith, 2017), Time Domain Neural Audio Style Transfer (Mital, 2017), Style Transfer for Musical Audio Using Multiple Time-Frequency Representations (Barry & Kim, 2018), and Style Transfer for stories in text form (Kiros, 2015).

DCGAN (2015)

Generative Adversarial Networks (GANs) were first proposed as a means to avoid having to explicitly formulate a loss function, and instead, learn such a function via an additional DNN (I. J. Goodfellow et al., 2014).

²⁴<https://neuralzoo.com/>

²⁵<https://chrisrodley.com/2017/06/19/dinosaur-flowers>

While Goodfellow et al demonstrated their first GAN on toy problems in 2014, it was again in 2015 that Radford et al demonstrated results that were incredible for the time, and caught the public’s imagination. Combining convolutional layers²⁶ with GANs, the researchers demonstrated the potential of **Deep Convolutional Generative Adversarial Networks (DCGAN)**. A DCGAN is a deep generative model, as we discuss in section 2.1: *Generative models*, that is able to capture the distribution of a dataset of images, such that a number of meaningful manipulations are possible in a space of compact latent representations. Radford et al’s DCGAN were relatively low quality, with image dimensions of only 64x64 pixels. Their model also suffered from typical GAN issues of the time, such as deformed outputs and instability during training. Despite this, the model did demonstrate many powerful latent manipulations as we describe in subsection 2.1.3: *Latent manipulations*. Inspired by this, research into GANs continued to intensify.

Some of the notable early adopters of *unconditional* DCGANs, in an artistic and creative context, are artists such as Anna Ridler, Robbie Barrat, Jake Elwes, Mike Tyka, and Gene Kogan, to name a few.

Within the Creative DL community today, *unconditional* DCGANs are undoubtedly the most popular method of producing visual content. Initially this was with Radford et al’s original DCGAN, later ProGAN (Karras et al., 2017), BigGAN (Brock et al., 2019), StyleGAN (Karras et al., 2019) and at this moment in time, StyleGAN2 (Karras et al., 2020).

Pix2pix (2016), CycleGAN (2017)

Another significant milestone came a year after Radford et al’s DCGAN, with a *conditional* GAN colloquially known as **pix2pix**, Image-to-Image Translation with Conditional Adversarial Networks (Isola et al., 2016). As we have described in subsection 2.1.2: *Conditional generative models*, pix2pix is a conditional GAN, where the conditioning input is an image. This allows the model to effectively transform an input image, into some kind of output image.

Pix2pix was shown to be very successful at learning various kinds of image mappings. These include Style Transfer-like transformations; *image analogies* (Hertzmann, 2001) as we mentioned above, transforming low-colour semantic maps into photographic detailed images; colouring black and white images; and transforming photographs between night and day, or different seasons.

For this reason, pix2pix also became quite popular among the early adopters of Creative DL. One of the famous early applications of pix2pix came in the form of an online *edges2cats* drawing application (Hesse, 2017). In this online app, users can sketch on-screen, and the application then tries to turn the simple line drawings into photorealistic images of cats. This further fuelled the popularity of pix2pix.

Pix2pix is also the architecture that we use in chapter 5: *Learning to see: Digital puppetry through realtime video transformation*. We develop a parametrisable, live video processing and transformation system. We implement a custom training pipeline that generates image pairs on the fly, greatly improving the Generator’s ability to generalise to novel input images. We also introduce a realtime image processing pipeline, that exposes *human-understandable* parameters

²⁶We discuss these in subsection 2.3.15: *Convolutional Neural Networks (CNN)*.

that can be manipulated continuously in realtime; and also transforms any input image to a space that allows the Generator network to further transform the image to a desired target image space.

One of the major constraints of Pix2pix, is that the training process requires hundreds, if not thousands, of *matching pairs of images*. In other words, the training data must consist of (*input, target*) image pairs, and the model tries to learn a mapping from the input image to target image. This is not always possible, and the following year, **CycleGAN** was released, addressing this issue by allowing *unpaired* image-to-image translation (Zhu et al., 2017).

Within the Creative DL community, while unconditional DCGANs and pix2pix are widely used, CycleGAN did not become as popular. Helena Sarin and Mario Klingemann are amongst notable artists who are using CycleGAN within their artistic practice.

AlphaGo (2016), AlphaZero (2018)

Another important milestone that came in 2016, is the super-human Go playing AI AlphaGo (Silver et al., 2016), which was later developed into the more diverse, super-human Go, Chess and Shogi playing AI AlphaZero (Silver et al., 2018).

Game-playing AI is not a topic that we generally touch upon in this thesis. And while this may not typically be considered a ‘creative output’, we include it in this section for a number of reasons. First, an AI system such as AlphaGo beating human Go champions such as Lee Sedol and Fan Hui, is a very important milestone both culturally, and technically.

Second, it can be argued, that AI systems such as AlphaGo and AlphaZero, do exhibit creative behaviour in their decision making process. This is reflected in countless commentaries by domain experts such as Go 9-dan pro Michael Redmond, chess grandmaster Garry Kasparov, and many others.

Third, which is perhaps the most relevant reason, is that one of the underlying methods used in AlphaGo and AlphaZero, is very similar to a method that we developed and use in one of our early studies.

This method involves combining a Monte Carlo Tree Search (MCTS) driven agent, with a Convolutional Neural Network (CNN), to evaluate the state value at any given point in time during a simulation rollout.

The purpose of our AI agent is not to play games per se. Instead, we use an agent-based approach to collaborative, generative sketching, and we discuss this in section 3.2: *Collaborative generative sketching with MCTS and CNNs*.

Sketch-RNN (2017)

The agent-based collaborative, generative sketching method that we mention above, was not able to produce the realistic results that we were looking for, and we discuss the reasons for this in more detail. However, a year after our work, researchers from Google released **Sketch-RNN** (Ha & Eck, 2017). This is both a paper; and an accompanying online, collaborative, generative sketching application²⁷. The researchers use a very different technical approach to

²⁷<https://magenta.tensorflow.org/sketch-rnn-demo>

ours, however the motivations that they address, and the application that they build, are the same as we describe in our study. Namely, this is to build a generative system that is able to model and sketch images not as dense, pixel-based representations, but as sparse vector representations. And most importantly, to allow for a human to meaningfully interact with the system while it is sketching. We discuss all of these points in more detail in section 3.2: *Collaborative generative sketching with MCTS and CNNs*.

Progressive Growing of GANs, ProGAN (2017)

Another key milestone came in 2017, when researchers from Nvidia released seminal research known as *Progressive Growing of GANs* (ProGAN) (Karras et al., 2017). Their progressively growing architecture — combined with immense GPU requirements — allowed them to overcome the long standing issues with DCGANs not being able to capture cohesive structure at larger scales. The researchers were able to produce very realistic looking images at much higher resolution and quality compared to previous DCGANs. In just two years, DCGANs went from generating low quality, 64x64 pixel resolution images in (Radford et al., 2015); to very high quality, 1024x1024 pixel resolution images. With this, the focus of mainstream research began to shift. Up until this point, the majority of research within GANs were focused on the engineering side of trying to improve the quality and stability of GANs. Once Nvidia demonstrated that GANs could indeed produce high quality images, more research teams started to shift into investigating ways of allowing users to *control what* GANs produced.

ProGAN is one of the architectures that we use as a base in the research that we present in chapter 6: *Deep Meditations: Latent storytelling*, where we investigate methods of story telling and Meaningful Human Control over narrative in the latent space of a generative model. As we discussed in section 2.1.3: *Latent walks*, typically the *videos* produced using an image based DCGAN such as ProGAN, are *random walks in latent space*. In other words, a number of points are sampled randomly within the latent distribution of the model, and then these points are interpolated, and a video is produced. Effectively, this video has a *random* narrative. The problem that we seek to address in our study, is to devise a method that allows *Meaningful Human Control over the narrative* of a story constructed in the latent space of an unconditional generative model, such as ProGAN. In this respect, our method is not strongly reliant on ProGAN, but can and does work with other unconditional generative image models such as BigGAN (Brock et al., 2019), StyleGAN (Karras et al., 2019) and StyleGAN2 (Karras et al., 2020). However, we primarily use ProGAN, since it was the state-of-the-art unconditional generative image model at the time that we conducted this study.

BigGAN (2018)

Despite the incredible image quality exhibited by ProGAN, this is generally limited to cases where the dataset is relatively constrained in terms of diversity. In other words, it is possible to successfully train a ProGAN, on a dataset such as CelebA-HQ (Karras et al., 2017), consisting of *tens of thousands of images* of faces perfectly cropped and aligned in the same manner. In this case, such a ProGAN model is able to generate very believable samples at image resolutions

as high as 1024x1024 pixels. However, incredibly diverse and complex datasets such as ImageNet (J. Deng et al., 2009), which consists of *millions of images* across thousands of different categories, still represent problems for ProGAN. This is to say that, ProGAN architecture is unable to model such a diverse dataset, and sampling from such a model, produces unrealistic images that do not represent the data distribution in terms of fidelity or variety.

In 2018, researchers at Google attempt to address this issue by investigating the impact of increasing scale. They leverage Google’s compute resources to increase the parameter count by four-fold, and the batch-size by eight-fold, compared to the state-of-the-art at the time. The resulting model, colloquially known as **BigGAN**, demonstrates great potential in its ability to capture the data distribution, both in terms of variety, and fidelity, outputting images at resolutions up to 512x512 pixels without the need for progressively growing architecture. When the researchers released pretrained models, this again captured the attention and imagination of many artists, designers, and researchers from different disciplines, and BigGAN experiments became wildly popular on social media²⁸, some of which were our own²⁹.

In addition to ProGAN, BigGAN is one of the architectures that we investigate in chapter 6: *Deep Meditations: Latent storytelling*. However, since training BigGAN is not possible with the hardware that we have access to, we test our methods on the *pretrained* model released by the researchers. This is in fact one of the motivations behind the method that we present. In that, we set out to devise a method that would work with pretrained models, when we do not have the possibility to train our own models.

GANBreeder (2018), ArtBreeder(2019)

BigGAN was also the catalyst for computational artist and designer Joel Simon’s wildly successful and popular website **ArtBreeder** (Simon, 2019). Simon was initially inspired by the earlier large-scale online collaborative evolutionary art app **Picbreeder** (Secretan et al., 2008). On Picbreeder, users can browse parametrically generated images, and select their favourites in order to *branch* them. The selected images undergo subtle random mutations, and they *evolve* to produce new images. Most crucially, Picbreeder is also an online community whereby users can view and select from the images generated by other users. Across many generations and many users, entirely brand new images are created in a massive online collaboration.

Simon realised that a very similar massively collaborative evolutionary approach could apply to generative deep models, due to the fact that they possess a latent code \mathbf{z} , which is in effect a very compact representation of the image, such that minor modifications to the latent code, results in minor modifications to the image generated. The first version of the online app came in 2018 with the name *GANBreeder*, and only supported the pre-trained BigGAN model as released by the researchers (Brock et al., 2019). Later renaming this to *ArtBreeder* in 2019, Simon added new StyleGAN and StyleGAN2 models trained on datasets such as portraits, anime, 3D characters, landscapes, and album covers. The current version of the website also provides the latent manipulation methods that we describe in subsection 2.1.3: *Latent manipulations*. All of these operations can be performed via a very simple-to-use online GUI, making the system

²⁸<https://twitter.com/search?q=%23biggan>

²⁹<https://twitter.com/search?q=from%3Ametotv+biggan>

very accessible. As a result, ArtBreeder is incredibly popular, and used widely by many people. According to the website, at the time of writing, over 72 million images have been generated using the online app.

StyleGAN (2019), StyleGAN2 (2020)

After ProGAN, researchers at Nvidia would go on to introduce two new architectures known as StyleGAN (Karras et al., 2019), and StyleGAN2 (Karras et al., 2020). In this research, they build upon their work in ProGAN to minimise undesirable artefacts in the images generated, as a result, producing even more realistic results. The biggest contribution however, comes in the form of an updated Generator architecture that learns to separate multi-scale attributes in an unsupervised manner. Inspired by the Style Transfer literature, StyleGAN and StyleGAN2 allow for the generation of random samples such that the *content* and *style* can be controlled independently. For example, when generating a sample of a random face from a model trained on portraits, we can sample a random face image, and then modify the high-frequency details such as hair or skin characteristics like wrinkles, freckles or colour, without affecting the underlying bone structure or pose. This is made possible via an additional embedding that the authors introduce known as w -space. This is an intermediate latent space whereby the original z -vector is fed through an additional mapping network f , to produce an array of w -vectors. As opposed to being conditioned on a single z -vector, as is typical in a DCGAN, each of the convolutional layers of the Generator is conditioned on a corresponding w -vector. As a result, manipulating multiple w -vectors instead of a single z -vector allows us to control the samples generated from the model in a much more precise manner.

ArtBreeder leverages this and provides a very simple to use GUI which allows users to mix between the *style* and *content* of different samples independently.

The introduction of the StyleGAN architecture in 2019, marks another important milestone in which *Meaningful Human Control* is, perhaps for the first time, becoming a prominent area of research within creative generative DL. And as mentioned, in recent years, this is an area that is now gaining more interest from DL researchers (Park et al., 2019; Karras et al., 2019; Simon, 2019; Bau et al., 2019; Karras et al., 2020; Härkönen et al., 2020; Jiang et al., 2020; Broad et al., 2020).

2.2.8 Computational Creativity

As is sometimes typical in academia, separate fields might exist which at first glance may appear to be covering the same topics, but in fact, are asking subtly different questions, and as a result exist as parallel fields of research. **Computational Creativity** is such a field of research which is important to mention in the context of *Creative DL*. While there is indeed plenty of overlap between the fields of *Computational Creativity*, and *Creative DL* as we discussed above, they exist as distinct fields.

Computational Creativity is a slightly more mature field in comparison to Creative DL, as it is not focused on the technology of DL, in fact it is not focused on any technology at all, but is based around more of a philosophical question (Boden, 1998).

Whereas the field of AI was born questioning whether a machine can *think*, or exhibit *intelligent* behaviour (Turing, 1950), Computational Creativity questions whether a machine can be *creative*, or exhibit *creative* behaviour (Jordanous, 2014).

Computational Creativity research is not only concerned with the creative output of the algorithms or technical implementation details, but is equally — if not more — concerned with the philosophical, cognitive, psychological and semantic connotations of *machines exhibiting creative behaviour*, or *acting creative*. In (McCormack & D’Inverno, 2012) and related papers (McCormack, 2014; McCormack et al., 2014), the authors ask — and attempt to answer — questions regarding computers and creativity, creative agency and the role of creative tools.

As part of the philosophical angle of Computational Creativity research, there is often an emphasis on *fully autonomous* systems. The field includes research into computational models of *serendipity* (Corneli et al., 2014), or software which exhibits *intentionality*, and is able to justify the decisions it makes when creating a piece of work by *framing* information in the context of the work (Colton & Wiggins, 2012). This can be thought of as analogous to an artist making deliberate, purposeful decisions at every step of the creative process. This aspect of computational creativity is sometimes referred to as *strong computational creativity* (Al-rifaie & Bishop, 2015) — analogous to John Searle’s *strong* (vs *weak*) AI (Searle, 1980). The field is also accompanied by certain formalisms, proposed models and theories of creativity to ensure the systems’ behaviour complies with what is thought to be ‘creative behaviour’ (Colton et al., 2011). Within this context there has been research into systems that conceive fictional concepts (Cavallo et al., 2013), design video games (M. Cook et al., 2014), write poetry (Colton et al., 2012), and other inherently ‘creative’ tasks.

We can summarise this by stating: the field of *AI* is interested in hypothesising, building and testing computational models of *intelligence*, while *Computational Creativity* is interested in hypothesising, building and testing computational models of *creativity*. In contrast, *Creative DL* is interested in developing and applying state-of-the-art DL techniques to the production of artistic works and creative media.

While we do find the research within Computational Creativity to be very intriguing, these discussions, particularly the formalisms and models of creativity, are not within the scope of our research.

It could be argued that our research is interested in *weak computational creativity* — particularly *semi-autonomous*, *collaborative creativity* where human interaction in the creation process is not only relevant, it is *essential* — to create interactive systems whereby the human user can guide the computationally creative system in realtime.

2.2.9 Machine Learning for Artistic, Expressive Human Computer Interaction (AE-HCI)

We feel that it is also important to take a moment to acknowledge the involvement of ML in the production of artistic works and creative media, prior to DL. This is a very rich field, especially within the areas of music and sound. Within these topics however, we will focus again on both Meaningful Human Control, and Realtime Continuous Control. One of the key areas that ML has contributed to within this intersection, is in the recognition and interpretation of *gestures*,

and more broadly speaking, aiding *Artistic Expressive Human Computer Interaction (AE-HCI)* — Human Computer Interaction *for artistic expression*.

In our research, we do not set out to design generative systems which allow for *embodied or gestural interaction* per se. We do however, take the principles that we describe below as strong guidelines, and we see embodied and gestural interaction as an end goal. In other words, in our research, we seek to introduce modes of interaction which could relatively easily be translated into embodied and gestural interfaces. We reflect on these end goals in the subsequent chapters when we present each of our studies, and the modes of interaction that we have developed for them.

In (Dourish, 2001) Paul Dourish proposes new models for interactive system design. *Embodied Interaction* is interaction embodied in the environment, not just physically, but as a fundamental component of the setting. It is an interaction design that takes into consideration the ways we experience the everyday world. This philosophy is particularly applicable when designing gestural interfaces for artistic expression.

As mentioned before, Myron Kruger was also interested in exploring *Responsive Environments*, in which ‘interaction is a central, not peripheral issue’ (Krueger et al., 1985). He saw potential in this area for the arts, education, telecommunications as well as general human-machine interaction and was motivated by creating playful environments which explore the perceptual process we use to navigate the physical world.

Human Computer Interaction for music — or *Musician-Computer Interaction (MCI)* (Gillian, 2011) — is one of the more academically established fields related to AE-HCI, more so than gestural Human Computer Interaction for *visual composition*. However, many of the requirements for interaction design and particularly gesture recognition are similar. Both require low-latency, realtime systems that can be configured on-the-fly. They need to be capable of detecting a wide range of gestures, some AE-HCI systems might concentrate on subtle finger movements, while others track whole bodies of multiple people. Furthermore, the ability to detect and respond to subtle variations in gestures is essential to convey expressivity (Caramiaux et al., 2014). Also, in performance situations, gesture recognition need not be *generalised across different people*, but training can be specific to the performing individual to maximise personal expression (Gillian, 2011).

Due to these similarities, in our research, Musician-Computer Interaction is taken as a base model for expressive gestural interaction, and will be built upon for general AE-HCI.

Gestures and ‘Expressive Gesture’

A survey of definitions of *gesture*, especially in relation to music can be found in (Cadoz & Wanderley, 2000). The authors conclude that the many proposed definitions do not adapt to *gesture in music*, but they purposefully avoid providing a new definition, focusing instead on which aspects of the various definitions might apply.

In (Camurri et al., 2004) the authors define *Expressive Gesture* as “responsible of [sic] the communication of information that we call expressive content” where “Expressive content concerns aspects related to feelings, moods, affect, intensity of emotional experience”. This is the definition of *Expressive Gesture* that is used in our thesis, complemented with the “natural,

spontaneous gestures made when a person is telling a story”, as described in (Cassell & McNeill, 1991). Particularly those with the semiotic classification of *metaphoric*, indicating abstract ideas (McNeill & Levy, 1980). A wider study of gesture expressivity and its dimensions — especially in the context of musical performance and Human Computer Interaction — can be found in (Caramiaux, 2015).

Gestural Interaction (and Gesture Recognition) for AE-HCI

Gestural interaction (and gesture recognition) is a very broad field. In this section, we focus on applications within AE-HCI, particularly in context of Machine Learning. Wider surveys can be found in (Mitra & Acharya, 2007; Gillian, 2011; LaViola Jr., 2014).

Artificial Neural Networks (ANN) are particularly useful for AE-HCI as they are able to map *m-dimensional input vectors* to *n-dimensional output vectors* with a learned non-linear function, allowing them to control complex parameter-sets simultaneously. This is especially useful in *regression* tasks, when manipulating continuous parameters of a generative visual or sonic model. They can be equally successful in *classification* tasks, for recognising gestures and triggering desired visual or sonic outputs. In 1992 Michael Lee et al used ANNs inside the MAX/MSP musical programming environment to investigate adaptive user interfaces for realtime musical performance (Lee et al., 1992). They were able to successfully recognise gestures from a number of devices including a radio baton and a continuous space controller. In 1993, Sidney Fels and Geoffrey Hinton used ANNs to map hand movements captured via a data-glove, to a speech synthesiser (Fels & Hinton, 1993). They achieved realtime results with a vocabulary of 203 gestures-to-words demonstrating the potential of Neural Networks for adaptive interfaces. Now, with many open-source implementations, and also integrated into Rebecca Fiebrink’s *Wekinator* (R. Fiebrink et al., 2009) and Nick Gillian’s *GRTGui* (Gillian, 2011), ANNs are widely used for creative gestural interaction.

Many other Machine Learning techniques have been used for gesture recognition, with different specific use cases. These include K-Nearest Neighbours, Gaussian Mixture Models, Random Forests, Adaptive Naive Bayes Classifiers and Support Vector Machines to classify static data; Dynamic Time Warping and Hidden Markov Models can be used to classify temporal gestures; Linear Regression, Logistic Regression and Multivariate Linear Regression can be used for real-valued outputs as opposed to classifying the input. A survey of Machine Learning techniques and applications for musical gesture recognition can be found in (Caramiaux & Tanaka, 2013).

As mentioned previously, in an artistic, performative context, detecting subtle variations of gestures is vital to conveying expressivity. In (Bevilacqua & Muller, 2005; Bevilacqua et al., 2009), Bevilacqua et al design continuous gesture *followers* that allow temporal gesture recognition in realtime while the gesture is still being performed. This algorithm returns *time progression information* and *likelihood*, enabling performers to alter speed and accuracy of the gesture to control parameters of their generative model.

In (Caramiaux et al., 2014; Caramiaux, 2015) Caramiaux et al develop systems that go beyond classification of the gestures, to characterise the *qualities* of the gesture’s execution. They use computational adaptive models for identifying *temporal*, *geometric* and *dynamic* variations on the trained gesture. Returning this information in realtime to the performer as they

are executing gestures, enables the performer to map the variations to parameters such as time-stretching samples, modulations, and volume or custom synth parameters.

In (Kiefer, 2014) Kiefer investigates the use of *Echo State (Recurrent Neural) Networks* (ESN) as mapping tools, to learn sequences of input gestures, and non-linearly map them to multi-parameter output sequences. The research concludes that ESNs demonstrate good potential in pattern classification, multi-parametric control, explorative and non-linear mapping, but there is room for improvement to produce more accurate results in some cases.

Interactive Machine Learning (IML)

As discussed above, ML offers very successful tools for the recognition of patterns and gestures. However *using* Machine Learning can be difficult and inaccessible to many, due to the technical knowledge and time required in building classifiers and setting up the signal processing pipeline (Fails et al., 2003).

Interactive Machine Learning (IML) is a family of human-in-the-loop techniques led by principles from Human Computer Interaction (R. A. Fiebrink, 2011).

While ML brings huge advancements to the fields of data analysis and pattern recognition, IML seeks to improve how ML systems can be used. Particularly, expanding its user-base from dedicated computer scientists and closely related disciplines, to a much wider audience. One of the ways in which this is made possible, is via a Graphical User Interface (GUI) front end to a ML backend, with data streamed live to and from the ML backend. In this case, the training and predictions can be performed in realtime, without writing any code making it a perfect choice for performance and AE-HCI.

Rebecca Fiebrink et al's previously mentioned *Wekinator* software released in 2009 is an example of such an IML system aimed at musical performance (R. Fiebrink et al., 2009). Using a GUI, users are able to setup, train and modify parameters of an ANN. The software also allows other applications — such as existing music software, visual software, or other custom generative software — to stream data to the Wekinator using a UDP-based protocol commonly used in inter-app and inter-device communications called Open Sound Control (OSC) (Wright & Freed, 1997). As the Wekinator receives this data, it runs it through a Machine Learning model and streams back predictions in realtime. Using this tool, artists, musicians, dancers, performers and researchers from other fields can train and map gestures to arbitrary outputs, such as notes, effects, images and sounds with little-to-none technical Machine Learning experience and programming.

Nick Gillian's *Gesture Recognition Toolkit* (GRT) from 2011 (Gillian, 2011) provides similar functionality but with more emphasis on the signal processing / gesture recognition pipeline. It lacks built-in input functionality such as webcam or microphone inputs, but has a number of built-in pre-processing, feature extraction and post-processing algorithms. Examples for these are Fast Fourier Transform, Principal Component Analysis, various filters, derivatives, dead zones and more. In addition to being an open-source application, the underlying codebase is released as a C++ framework allowing it to be integrated into bespoke applications.

Tools like these enable both technical and non-technical users to quickly setup, train and test models for gesture recognition and gestural interaction. Without writing any code, users

can start streaming input data from their sensors, and receive predictions in their application of choice, enabling them to gesturally create, manipulate and perform audio-visual media in realtime. An example of Fiebrink’s Wekinator can be seen in the band 000000Swan’s audio-visual shows gesturally driven using a Microsoft Kinect and commercially available sensor bow (Schedel et al., 2011). In addition, it has also been applied in contexts such as workshops with people with learning and physical disabilities (Katan et al., 2015).

2.3 Introduction to Deep Learning

In this section, we take a slightly more technical approach to Deep Learning. We introduce key concepts, concentrating on those that are most related to our research. Where relevant, we provide theoretical and mathematical foundations, which we will build upon in later chapters. We do not present the developments in a chronological order, neither do we discuss parallel or orthogonal developments. Instead, here we focus on only the technical foundations that will be necessary for the rest of this thesis. Where relevant, we also highlight key issues which we will address in later chapters. For more comprehensive technical explanations, please see I. Goodfellow et al. (2016), and for a thorough historic survey of Deep Learning, please refer to Schmidhuber (2015).

2.3.1 Overview

Deep Learning (DL) is a branch of Machine Learning (ML), which in turn is a branch of Artificial Intelligence (AI).

We will unpack these ideas more formally in the following sections. However, for the purposes of this introduction, we can loosely define **Machine Learning**, especially as it relates to our research, as: *designing behaviours through examples* (R. A. Fiebrink, 2011; Penny, 2017).

When we wish to design a system that performs a specific behaviour, a typical approach might be to explicitly implement the behaviour. In order to do this however, we would first need to identify exactly how the behaviour functions. For a very complex behaviour, we would need to break it down step by step, to the degree that we can implement it. Unfortunately, there are many situations where we simply might not know how a desired behaviour functions. For example, predicting which player is more likely to win during a game of Go, by looking at the state of the board, is not a behaviour that we currently know how to explicitly define. There are also many situations, where we might *tacitly* be aware of the behaviour, without being consciously and explicitly aware of the steps required to replicate it. For example, we can often easily recognise objects in images, and words in sounds. Yet, we might find it impossible to explicitly formulate behaviours which can identify those exact same objects and words.

In these situations, we can use Machine Learning algorithms to implement those behaviours, by *providing examples of the desired behaviour*.

In other words, the desired behaviour can be thought of as a function f , which maps an input \mathbf{x} to an output \mathbf{y} . We can think of a non-ML approach, as manually identifying and implementing the function f . Whereas a ML approach, consists of providing examples of \mathbf{x} and \mathbf{y} to some *learning algorithm*, and the algorithm figures out what the function f should be.

Deep Learning is the application of this approach to very complex and high-dimensional data — such as images with millions of pixels or sounds with tens of thousands of samples per second — via *deep* parametrisable computation graphs that learn hierarchies of representations (LeCun, 2014; I. Goodfellow et al., 2016). In essence, these algorithms are capable of identifying complex relationships in vast amount of Big Data. They can recognise patterns and find regularities in data, that either we are aware of but unable to explicitly formulate, or we are not aware of at all. Thus, using DL enables us to create systems which exhibit more intricate and complicated behaviour than we would otherwise be able to implement.

Furthermore, even though the behaviour of these systems will be statistically consistent with the examples that we provide it, it is very probable that they will uncover patterns that we were *unaware* of. As a result, they are very likely to exhibit *unexpected behaviour*.

This is a quality of DL which is integral to our research. Our goal is to try and *exploit* and *tame* this unexpected behaviour, allowing human users to *experiment with* and *explore* this new terrain, while simultaneously providing enough meaningful control, such that they can also creatively express themselves, as opposed to producing entirely ‘random’ outputs.

2.3.2 Machine Learning (ML)

More formally speaking, ML is a field of Artificial Intelligence that investigates computational systems that can *improve their performance*, on a *particular task*, as measured by a *specific metric*, as they gain more *experience* with that task (Mitchell, 1997). In this context, the term *experience*, refers to observations, examples, or in other words: *data*.

ML algorithms build **models** based on observations, and they effectively learn rules, or functions, to make optimal decisions or predictions. There are a very large number of different approaches that fall within the scope of ML. A very in-exhaustive list includes algorithms such as linear regression, logistic regression, polynomial regression, support vector machines, K-means, expectation maximisation, genetic algorithms, markov chains, decision trees, random forests, naive bayes, bayesian networks, Artificial Neural Networks, and many many more. Each of these methods, have qualities that make them more or less suitable to a particular problem. For example, we would choose a particular approach, depending on the complexity of the problem, the nature of the expected solution, and the number of data points available.

2.3.3 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) are just one of these different approaches to Machine Learning. An ANN consists of a **graph**, a network of **neurons** (McCulloch & Pitts, 1943), very loosely inspired by biological Neural Networks. A numeric **weight** value is assigned to every connection in the network. Each neuron performs a weighted sum of all of its inputs, effectively a linear transformation, and optionally applies some **non-linear activation function** (such as *tanh*, *sigmoid*, *ReLU* etc), and then passes the result onto the next neuron in the network for a similar transformation.

A Neural Network acts as a function f , that when given an input \mathbf{x} , computes an output \mathbf{y} . The structure and shape of the network, and the activation functions used on each neuron, define the **architecture** of the ANN. The function f that the ANN computes, is defined by its

architecture, and collectively, all of the **weights** of each connection. The weights of the ANN are also known as its **parameters**, and is typically denoted by Θ . In other words, we can define the function computed by the ANN as $f(\Theta) : \mathbf{x} \mapsto \mathbf{y}$. We will see in later sections, that the act of *training* an ANN, involves solving for the parameters Θ in order to obtain a desired function.

2.3.4 Feed-forward (FNN) vs Recurrent Neural Networks (RNN)

When there are no *cyclic connections* in the graph, and data always flows in one direction, then the ANN is called a **Feed-forward Neural Network** (FNN). When there are cyclical connections, it is called a **Recurrent Neural Network** (RNN). RNNs allow the Neural Network to model *temporal or sequential* data. We will examine RNNs in more detail in subsection 2.3.19: *Recurrent Neural Networks (RNN)*.

2.3.5 Layers

Typically, ANNs are neatly arranged in **layers**, as opposed to consisting of more unorganised layouts as is more common in *biological* neural networks. This makes it simpler to mathematically model and computationally implement ANNs, especially when using highly parallelised *Graphics Processing Units* (GPU) or *Tensor Processing Units* (TPU). Each layer — which might consist of thousands or even millions of neurons — can be treated as a *single, high-dimensional, multi-variate unit*. The ANN can be thought of as simply a chain of *layers* (or *units*). Each layer receives a single *vector* from the previous layer in the chain, and outputs a single vector to the next layer in the chain. The collection of *weights* between two adjacent layers, can be represented as a **weights matrix**. We can also clearly define the **input layer** as the first layer in the ANN which receives the inputs as a vector, the **output layer** as the results of the computation performed by the ANN, and any layers in between are called **hidden layers**.

2.3.6 Multi-Layer Perceptrons (MLP)

When an ANN is arranged in fully connected layers, with no cyclic connections, then the ANN is known as a **Multi-Layer Perceptron** (MLP) (Rumelhart et al., 1985). A simple example of this can be seen in Fig. 2.4.

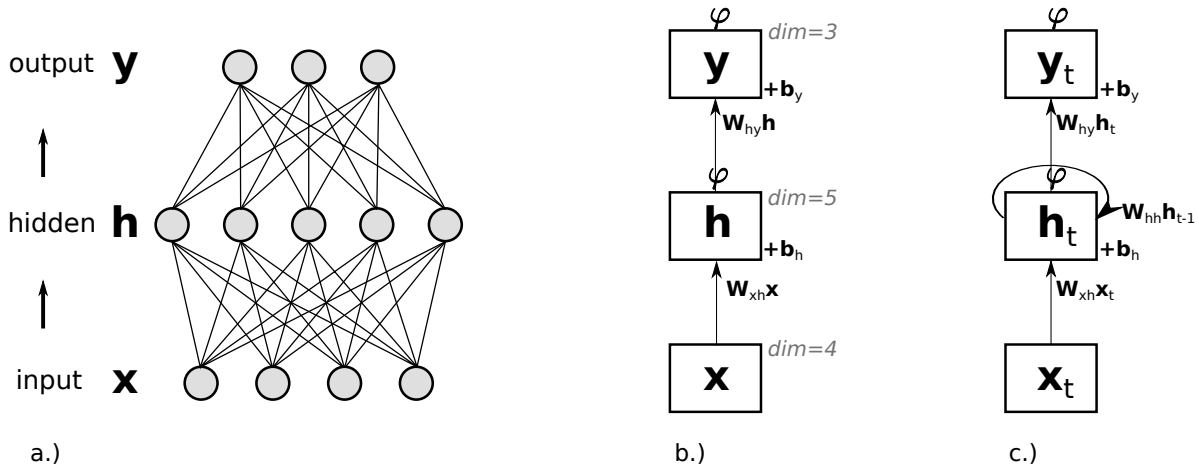


Figure 2.4: a.) A three layer, feed-forward, Multi-Layer Perceptron with a single hidden layer. In this diagram, all of the neurons are displayed, and we can see that the network consists of 4 inputs and 3 outputs, and a 5 neuron hidden layer. b.) The same MLP, displayed with layers collapsed into multi-variate *units* allowing simpler visualisation, mathematical modelling and computational implementation using vectors. The dimensions of each layer is indicated with *dim*, \mathbf{W}_{xh} and \mathbf{W}_{hy} denote the weights of the connections from the input layer to the hidden layer, and hidden layer to the output layer respectively, \mathbf{b}_h and \mathbf{b}_y denote the bias vectors (which are typically included as parts of the weights or parameters) of the hidden layer and output layer respectively, and φ denotes the activation functions. c.) A Recurrent Neural Network, with a cyclic connection on the hidden layer.

Each layer of a MLP can be thought of as a multi-variate *unit*, which transmits a vector to the next layer, where the value of each layer is a function *only of the previous layer's output*. For a MLP with a single hidden layer such as the network presented in Fig. 2.4, this can be formulated as

$$\mathbf{h} = \varphi_h(\mathbf{W}_{xh}\mathbf{x} + \mathbf{b}_h), \quad (2.3)$$

$$\mathbf{y} = \varphi_y(\mathbf{W}_{hy}\mathbf{h} + \mathbf{b}_y), \quad (2.4)$$

where $\mathbf{x}, \mathbf{h}, \mathbf{y}$ are respectively the *input*, *hidden* and *output* vectors. We adopt the notation \mathbf{W}_{jk} and \mathbf{b}_k to respectively denote the weights matrix from layer j to k and the bias vector for layer k , where the subscripts x, h, y respectively denote the *input*, *hidden* and *output* layers. φ_k is an element-wise (usually non-linear) activation function for layer k such as tanh, logistic sigmoid where $\varphi_k(x) = 1/(1 + e^{-x})$, or ReLU where $\varphi_k(x) = \max(0, x)$ (Nair & Hinton, 2010) or one of many others. This architecture forms the foundation for all of the architectures that we present in the following sections.

2.3.7 Universal function approximators, expressive power

When non-linear activation functions are not applied to any of the layers, then the entire MLP effectively acts as a linear transformation — which is not desirable if the data that the ANN is trying to model is non-linear. However, when non-linear activation functions are applied, then it has been shown that even a three-layer MLP, with a single hidden layer, can approximate any continuous function to an arbitrary desired level of accuracy (Hornik, 1991). For this reason, such Neural Networks are known as **universal function approximators**.

However, not *any* three-layer MLP can model *any function*. The **expressive power** of a Neural Network, is defined by its architecture (Lu et al., 2017). This poses one of the key challenges when working with ANNs, in that we must design a Neural Network architecture such that it provides sufficient expressive power for the data that we are trying to model.

Furthermore, in the context of *learning*, designing an architecture which provides sufficient expressive power is not our only challenge. Even in the situation that an ANN's architecture *does* provide the expressive power required to model a particular function, that does not guarantee that the ANN will be able to *learn* that function, especially from the data that we have available. We will explain the reasons in the following sections.

Note that in the case of an MLP, the architecture of the network can be simplified to thinking of the **depth** of the network (how many layers the network consists of), and its **width** (how many neurons are in each layer). These architectural variables, are collectively referred to as **hyperparameters**. (This term, helps distinguish them from the *parameters* of the Neural Network, which are its *weights* Θ).

2.3.8 Learning

As we mentioned previously, the function f that an ANN will compute, is determined by the network's *architecture*, and its *weights* Θ . Within the context of ML, we generally do not know what this function is. Instead, we have observations, i.e. **training examples**, which we wish to fit a function to. Typically, we construct an ANN with a pre-defined architecture, which we select based on experience (this is not always the case, we will expand on this in later sections). We then treat the weights of the ANN, as unknown **parameters** which we wish to solve for. We then solve for the weights of the network, such that the function that the network ultimately computes, fits the training examples.

In other words, we assume there is a function g , that governs the behaviour that gives rise to our training examples, the observations that we have made. We can think of this function g , as the **ground truth**. Our goal is to find the weights Θ of an ANN, such that $f(\Theta) \approx g$.

In this respect, we can frame **learning**, as a **search** problem. It is the process of finding the weights Θ , such that the outputs computed by an ANN when it receives some training examples, are as close as possible to the ground truth. In other words, **training** an ANN, typically consists of trying to find the weights Θ that **minimize some error** between $f(\Theta)$ and the ground truth g .

2.3.9 Loss functions

For this, we define a **loss function** \mathcal{L} , also known as a **cost**, **error** or **objective** function. This function will compute a quantitative metric for how close $f(\Theta)$ approximates g . We can construct many different loss functions, depending on the nature of the problem that we are looking to solve, and the data that we are trying to model.

For example, given \mathbf{y} , the output of an ANN for some given input \mathbf{x} , and $\hat{\mathbf{y}}$, the true value of the observation for the same input, some common loss functions that we use in our research can be formulated as

$$\mathcal{L}_{MSE} = (\mathbf{y} - \hat{\mathbf{y}})^2 \quad (\text{Mean Squared Error or } L2 \text{ loss}) \quad (2.5)$$

$$\mathcal{L}_{MAE} = |\mathbf{y} - \hat{\mathbf{y}}| \quad (\text{Mean Absolute Error or } L1 \text{ loss}) \quad (2.6)$$

$$\mathcal{L}_{CE} = -(\hat{\mathbf{y}} \log(\mathbf{y}) + (1 - \hat{\mathbf{y}}) \log(1 - \mathbf{y})) \quad (\text{Cross Entropy or Log loss}) \quad (2.7)$$

Unsurprisingly, the loss function that we select, has a very significant impact on how quickly and efficiently the ANN learns, and the solution that it converges to. We discuss this in more detail in later sections, particularly in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, where we present a software tool that we have developed, that performs *realtime training* on a live video feed. Our software allows us to select a loss function from a number of different options via a graphical user interface in realtime, while the network is training, and we can immediately observe the impact of the different loss functions.

Having defined a loss function, we can now formulate the act of learning, as an **optimisation** problem, and there are a number of methods for solving this. Stochastic, Monte Carlo methods and evolutionary algorithms are a few common examples. However, the most popular methods in use today, are based on **gradient descent and backpropagation**.

2.3.10 Gradient descent and backpropagation

Gradient descent, is an optimisation algorithm that tries to minimise some function, by iteratively following the function's *gradient*, the *path of steepest descent*. To be able to train an ANN using gradient descent, the loss function that we wish to minimise, and the entire chain of transformations as defined by the Neural Network, has to be *end-to-end differentiable*. The typical ANNs that are most popular today, including those that we use in our research, are as we described above: layered sequences of linear transformations, followed by simple non-linear activation functions. For this reason, they are indeed end-to-end differentiable.

The training process can then be summarized as: i) iteratively feed one or more training examples to an ANN to produce some outputs, ii) calculate and differentiate the loss with respect to the parameters Θ to compute the **gradient** of the loss, a vector pointing in the direction of steepest descent, and finally iii) run a **backwards pass** through the network, and **backpropagate** the error. In other words, for every neural connection, compute the amount by which we should *nudge* each weight, such that the overall loss will move in the direction of the gradient. Running this process over many iterations — usually in the order of millions — optimally nudges the weights such that $f(\Theta)$ approaches g .

A more detailed explanation is beyond the scope of this introduction, as we do not deviate from a typical gradient descent and backpropagation in this thesis. However, we will expand on this a little bit when we discuss training RNNs in section 2.3.19: *Backpropagation Through Time*, and again when we present our realtime training tool in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*. For more details on gradient descent and backpropagation in general, please refer to (I. Goodfellow et al., 2016).

2.3.11 Optimisation algorithms

Some of the typical challenges involved in an optimisation process as we describe above, is to find the global minimum as quickly as possible, without getting caught in local minima. There are many **optimisation algorithms** designed to tackle these issues, such as stochastic gradient descent, momentum, adagrad, adadelata, rmsprop, adam, adamax, nadam etc. And there are many variables which control the behaviour of these optimisers, such as learning rate, momentum, gradient clipping thresholds and numerous implementation specific variables. These are collectively included within the **hyperparameters** that we have already defined.

Just as we discussed with the loss function, the optimisation algorithm and the hyperparameters that we use, have a very significant impact on how quickly an ANN learns, and the solution that it converges to. For this reason, we developed the software tool that we present in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*. This software allows us to change not only the loss function in realtime during training, but also select between different optimisation algorithms, and adjust their hyperparameters in realtime too, allowing us to observe the results immediately.

2.3.12 Deep Neural Networks (DNN)

We have already established that even though a simple three-layer MLP with a single hidden layer is a universal function approximator, this does not imply that *any* three-layer MLP can necessarily approximate *any* function, of *any complexity* (Hornik, 1991). The expressive power of an ANN comes from its architecture. Furthermore, even if an ANN's architecture does provide sufficient expressive power to compute the function g which underlies some observations, it is not guaranteed that the ANN will be able to *learn* that function from the available training examples (Lu et al., 2017). In other words, in the context of learning, *efficiency* becomes key.

For example, a Neural Network with an architecture A_1 might have one *thousand* parameters, while a much larger second Neural Network with an architecture A_2 might have one *billion* parameters. It is very plausible that both of these architectures are perfectly capable of expressing some function g which underlies some observations. In other words, there exists a set of one thousand parameters Θ_1 such that $f_1(\Theta_1) \approx g$, and a set of one billion parameters Θ_2 such that $f_2(\Theta_2) \approx g$. However, in the case that we are given only 500 training examples, it is potentially more likely that training the network with architecture A_1 will converge to a solution yielding the set of one thousand parameters Θ_1 that approximate g . While training the network with architecture A_2 might not converge to an accurate solution.

For this reason, architecture design and *hyperparameter selection* becomes crucial.

Deep Learning, is the name given to a family of methods, designed to tackle this issue.

A **Deep Neural Network** (DNN), is an ANN as we describe above, with many layers, in other words, *with a large depth*. In this case, each layer can be thought of as a parametrised, high-dimensional, non-linear transformation. And a DNN is then a chain of these transformations, that collectively learn a hierarchy of representations with varying levels of abstraction, in order to model some complex, high-dimensional data.

Deep Learning research, is primarily the study and exploration of different types of deep architectures for different types of tasks, and the accompanying algorithms and techniques that relate to the performance of these architectures, such as designing, training, testing, scaling, deploying etc.

There are many architectures studied within the framework of Deep Learning. Some of the architectures that we will cover in the following sections are: Convolutional Neural Networks (CNN), Variational Auto-Encoders (VAE), Deep Convolutional Generative Adversarial Networks (DCGAN), Pix2pix and Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNN).

2.3.13 Hyperparameter search

We have already described **hyperparameters**, as variables that drastically impact *how* and *what* a Neural Network learns. These include variables that relate to the *architecture* of the Neural Network, for example the number of layers, and the number of neurons in each layer. They also include the activation functions that are used on each layer. In addition, there are hyperparameters that relate to the *optimisation process*. For example the optimisation algorithm that is used, whether it be stochastic gradient descent with momentum, adagrad, adam, or one of many other algorithms. Furthermore, there are also the hyperparameters of these optimisation algorithms, such as learning rate, momentum, gradient clipping thresholds and many others. In later sections, we will see even more hyperparameters, such as the *filter depth* of Convolutional layers (subsection 2.3.15: *Convolutional Neural Networks (CNN)*), the bottleneck size in Auto-Encoders (subsection 2.3.16: *Auto-Encoders (AE)*), the regularisation weight in Variational Auto-Encoders (subsection 2.3.17: *Variational Auto-Encoders (VAE)*), and many more.

Learning these hyperparameters through an automated process, is currently not possible, although it is a very active area of research known as **meta-learning**, or **learning to learn** (Andrychowicz et al., 2016; Finn et al., 2017; Zoph et al., 2018; Rusu et al., 2019). Instead, optimal hyperparameters must be selected manually. And selecting hyperparameters, as optimally suited for the particular dataset and problem that we wish to address, is currently still a magical art. There are some guidelines on how to select some of the hyperparameters, based on previous experience, if the nature of the dataset and the problem that we are dealing with is similar to a previous problem that has already been solved. However, in general there are no clear rules as to how to select any of these hyperparameters.

For this reason, it is not uncommon to train *hundreds* of models, varying the hyperparameters slightly in each case, in order to find the optimal configuration of hyperparameters that result in a model that converges to a desirable solution. This is known as **hyperparameter search**. The hyperparameters can be varied by fixed intervals in each training session, resulting in a *grid search*, or they can be varied randomly, resulting in a *random search* (Bergstra & Bengio, 2012).

Addressing this problem, is one of the key motivations behind the work that we present in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, where we develop a software tool that trains on a live video feed in realtime, while allowing us

to modify any of the hyperparameters during the process, and immediately observe the results.

2.3.14 Classes of learning

Before we discuss deep architectures in more detail, we will first describe different modes of learning.

Supervised Learning

In **supervised learning**, a model is trained on *labelled* data where each training example is an $(input, target)$ pair. During training the learning algorithm tries to find the parameters of the model Θ which effectively implement the function that maps the *input* of each training pair to the associated ground truth *target*. For **classification** problems the *target* is usually a discrete class label, often represented as a **one-hot vector** (i.e. a vector where all elements are zero, except for the entry for the desired class, which is one). For **regression** problems the *target* is usually a real-valued vector (or tensor). Having $(input, target)$ pairs makes it relatively straightforward to specify the objective function, as we simply need to calculate some kind of a difference between the model's output and the ground truth target, as we described in subsection 2.3.9: *Loss functions*. For this reason, supervised learning is currently one of the more popular and successful branches of ML. However, the training pairs often need to be manually associated by people. This makes them cumbersome and very time consuming to prepare. Online crowdsourcing platforms such as Mechanical Turk have helped accelerate the preparation of large labelled datasets which is one of the reasons why we're starting to see more success in this field (LeCun, 2014).

Unsupervised Learning

In **unsupervised learning**, training is performed on *unlabelled* data. Without an external supervisory signal it can be more ambiguous as to how to specify an objective function. For this reason, unsupervised learning is currently one of the big open problems in ML. A common objective in unsupervised learning is *dimensionality reduction*, including methods such as t-SNE (T-distributed Stochastic Neighbor Embedding) (Laurens van der Maaten & Geoffrey Hinton, 2008) or UMAP (Uniform Manifold Approximation and Projection) (McInnes et al., 2018). Another common objective is *clustering*. In this case, the learning algorithm tries to organise observations into groups based on similarities and regularities that it tries to identify across the entire dataset. In this thesis, we use two other types of unsupervised learning algorithms which we discuss in subsection 2.3.17: *Variational Auto-Encoders (VAE)* and subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*.

Semi-supervised Learning

A combination of the two — **semi-supervised Learning** — is used when some of the data is explicitly labelled, while some of it is not. In this case, during training, the learning algorithm tries to learn regularities in the data in an *unsupervised* manner, that allow it to cluster and

classify the data. With this knowledge, the algorithm can then associate the unlabelled items with the correct labels based on the labelled data, and then proceed in a *supervised* manner.

Reinforcement Learning

Reinforcement learning (RL) is technically and conceptually quite different to the above classes of learning, as it is rooted in a different discourse, and therefore does not necessarily use the same language as Neural Networks. RL is a type of learning that is typically applied to an **agent**, operating within a **Markov Decision Process** (MDP) (Bellman, 1957), where there is a notion of time, and at each **timestep**, the agent can take **actions** and move between different **states**. The actions that the agent takes, is governed by a **policy**, and that policy can be controlled by anything ranging from a simple behaviour, to a complex DNN. The agent regularly receives **reward signals** from the **environment**. However, such a reward signal might not be an *immediate* reward for the most recent action taken by the agent, but instead might be a **delayed reward** for an action — or even a series of actions — taken by the agent much earlier on. Alternatively, the reward might even be related to ‘random’ events outside of the agent’s control or knowledge. Therefore, a major challenge in RL, is to solve the **attribution problem** of these delayed rewards. For this reason, RL is considered neither supervised (with a direct supervisory signal) nor unsupervised (with no supervisory signal). Instead it is a form of learning based on a delayed reward signal (Kaelbling et al., 1996). The general objective of a RL algorithm, is to learn a policy that *maximises the agent’s long term reward*. This process involves a balance between **exploration** of new actions which have not yet been made, vs **exploitation** of actions which are known to reward higher than others. For this reason, RL can also be thought of as *learning by trial and error*.

In the following chapter section 3.2: *Collaborative generative sketching with MCTS and CNNs*, we present a method that we developed for interactive collaborative generative sketching with a Deep Neural Network, driven by Monte-Carlo Tree Search (MCTS). While MCTS is not considered typical RL, it has many similarities, and we unpack many of the concepts that we described in the previous paragraph in subsection 2.3.20: *Monte Carlo Tree Search*.

2.3.15 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are Neural Networks which consist of, and can learn, stacks of parametrisable *convolution filters* (LeCun et al., 1989; LeCun & Bengio, 1995). In **Convolutional layers**, neurons from one layer are not fully connected to all neurons in the next layer as they are in a typical MLP. Instead, *small patches* of neurons from one layer are connected to a *single* neuron in the next layer such that the operation performed by this group of connections acts like a **convolution filter** on that patch of neurons. Furthermore, this same pattern of connectivity with identical connection weights, is shared across the entire layer in a process known as **weight sharing**. This ensures that the *same convolution filter is applied across the entire layer*. In addition, this also ensures that the number of weights of the Neural Network is *radically smaller* compared to a Neural Network with similar connectivity that is not using weight sharing. This in turn radically aids training, encouraging convergence to a desirable solution in a much more efficient manner.

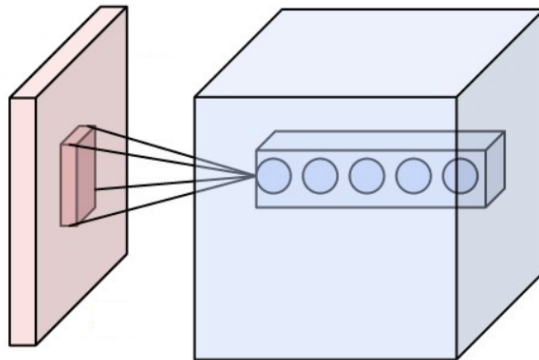


Figure 2.5: A Convolutional layer consisting of many filters, applied to an input. Image from Wikimedia Commons, licensed under CC BY-SA 4.0

CNNs are typically used for processing *pixel-based raster images*, and for this reason, they form the foundation for many of the architectures that we use in this thesis, and present in the following sections. Raster images are represented as a 2D grid — or *matrix* — of values. Greyscale images are represented as a matrix of real values. While colour images are represented as a matrix of *(red, green, blue)* tuples, or in other words, a *tensor* of rank 3, and dimensions $(w, h, 3)$ where w and h are the width and height of the image, respectively. For this reason, we generally think of the *inputs* and *outputs* of Convolutional layers to be *tensors*, as opposed to the one-dimensional *vectors* that we think of in a more typical MLP. We also do not think of the neurons as arranged in *linear* layers, as they are in a typical MLP. Instead, we think of the neurons arranged in layers of *grids*, with small **windows** of connectivity between layers that are replicated across the width and height of each layer.

For example, in a single Convolutional layer which consists of a **convolution window size** of 3×3 , only 9 weights are required to represent this filter across its entire input. However, if weight sharing were not used, this pattern of connectivity would require 9 weights *per input neuron*. In other words, if this convolutional layer were applied to an image with a width and height of 1024 pixels, it would require $9 \times 1024 \times 1024 \approx 9.4$ million weights instead of just 9!

CNNs have also been used in one dimension, to model raw audio (van den Oord, Dieleman, et al., 2016); and in three dimensions, to model 3D geometry using voxel representations (Z. Wu & Song, 2015). However, in this thesis we are primarily concerned with 2D images. For this reason, we focus on 2D CNNs.

At each Convolutional layer, small patches of the input are processed and condensed to a single value. For this reason, we can think of each Convolutional layer, as applying some learnt **filter** to its input, to extract some kind of **feature**. In addition, each Convolutional layer usually consists of many independently parametrised filters. Often called the **filter depth** of the Convolutional layer, the number of filters in a single layer is typically in the range 128 to 1024. As a result, applying such a Convolutional layer, simultaneously produces a large number of features. More formally, applying a Convolutional layer with a filter depth equal to d , to an image of dimensions (w, h) , results in a tensor of dimensions (w, h, d) . This can be seen in Fig. 2.5.

Across many layers, this results in an increasing **effective receptive field**, whereby a *single*

neuron in some high layer, will have indirectly processed, and will contain information from, a very large patch of an input many layers down in the hierarchy of the Neural Network, perhaps even the *entirety* of the original input image.

In addition, it is often very useful to reduce the dimensions of the input. For example, in a discriminative model designed to classify images and assign probabilities based on a number of class labels, we require an architecture which reduces an image consisting of millions of pixels down to a handful of class probabilities. Or in an *Auto-Encoder*, which we discuss in the following section, we require an architecture that compresses a large image down to a narrow bottleneck. In these cases we apply some kind of **subsampling** operation. Traditionally this was a **max-pool** operation whereby the maximum value of a small window replaces the entire window. A typical window size in this case might be 2×2 pixels. This ensures that every time the max-pool is applied, the layer size halves in width and height. For example, 8 max-pool layers interspersed throughout a CNN with many Convolutional layers, can reduce an image of dimensions $(256, 256)$ pixels, down to dimensions of $(1, 1)$, by incrementally halving it at every stage. As we previously mentioned however, each convolution layer is typically a *stack* of many parametrised filters, for example with a depth of 512. For this reason, the output of such a CNN will in fact be a tensor of dimensions $(1, 1, 512)$, where every value in this tensor each represents some different, independent piece of information *about the entire image*. This can be seen in Fig. 2.6.

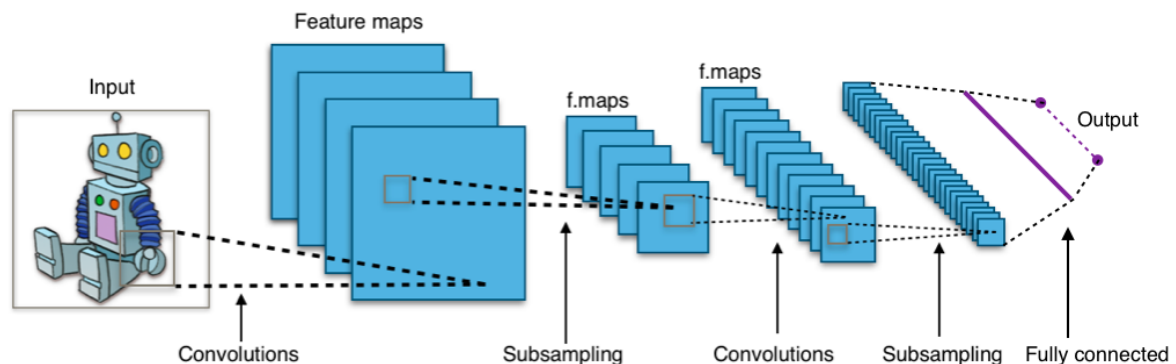


Figure 2.6: Image from Wikimedia Commons, licensed CC under BY-SA 4.0

A more modern, equally effective and computationally more efficient method of reducing dimensions in this manner is to use **strided convolutions**. In this case, instead of centring the convolution filters on *every* pixel in an input layer, a number of pixels are skipped. Typically, the filter is centred on every *other* pixel, in both horizontal and vertical directions. This again halves the resolution at every application of the strided convolution. Generally, the size of the convolution windows are in the order of 3×3 , 5×5 , 7×7 etc. For this reason, the information in the pixels that are skipped are not ignored by the filter, as they are still included within the convolution windows of the neighbouring pixels. However, the convolutions are never *centred* on these particular pixels. This cuts down the processing time to one quarter, while simultaneously providing the desired dimensionality reduction.

Depending on the desired use case of the CNN, a number of fully connected layers might also be added after all of the Convolutional layers. For example, we can consider the case of

a discriminative model designed to classify images. Using a number of strided Convolutional layers we can reduce the dimensions of an input image as we described above to a tensor of dimensions $(1, 1, d)$, where d represents some arbitrary filter depth, such as 512. We can then treat this tensor as a *linear layer*, and fully connect it to one or more linear layers, as we would in a typical MLP. The final layer of the MLP would then output the class probabilities.

In fact, this highlights one of the greatest strengths of Deep Learning. In such an architecture, we can think of the MLP portion of the Neural Network as a simple three or four layer MLP that takes some *compressed, abstracted, high-level representation* of an image, and predicts class probabilities. The Convolutional layers leading up to the MLP are effectively an image processing pipeline that outputs those compressed, abstracted, high-level representations. The key point however, is that the entire chain consisting of all of the Convolutional layers and the MLP, is end-to-end differentiable and learnt in single training session. This avoids the potentially problematic hand-crafted feature engineering phase that we discussed in the previous chapter under section 1.2: *Why Deep Learning?*

Since most of our work involves images, we often use Convolutional Neural Networks. More specifically, we use a number of different architectures, arranged in various different ways, which make use of Convolutional layers. We discuss these in the following sections.

2.3.16 Auto-Encoders (AE)

An **Auto-Encoder** is a Neural Network that tries to accurately reconstruct its inputs while passing data through a **bottleneck**. A simple example of such an architecture can be seen in Fig. 2.7. While this shows a very simple Auto-Encoder, with layers arranged as one-dimensional vectors, the layers can instead be Convolutional layers arranged in grids, as we describe in the previous section. Given a domain of inputs that contain significant amounts of structure and regularities, such as images of a particular category, we can for example successfully build and train Convolutional Auto-Encoders that compress colour images with width and height of 256 pixels ($256 * 256 * 3 = 196,608$ values) through a bottleneck of 128 dimensions, giving a compression ratio of 1536:1.

In order to successfully achieve this compression and reconstruction, the network has to learn how to compress the data in a meaningful manner. It has to learn how to build more concise and optimal representations, by identifying and exploiting salient features and regularities that occur in the data that is presented to it. Since no additional labels or training pairs are provided, training an Auto-Encoder is considered *unsupervised learning*.

An Auto-Encoder can be thought of as two Neural Networks back-to-back. The **encoder** is the portion of the network that learns a function E , which receives an input \mathbf{x} and maps that to a compressed, latent representation \mathbf{z} in the bottleneck layer. The **decoder** is the portion of the Neural Network that learns a function D , which maps a compressed, latent representation \mathbf{z} , to an output \mathbf{x}' . We use a loss function to ensure that $\mathbf{x}' \approx \mathbf{x}$. For example, using L2 loss we can formalise this as

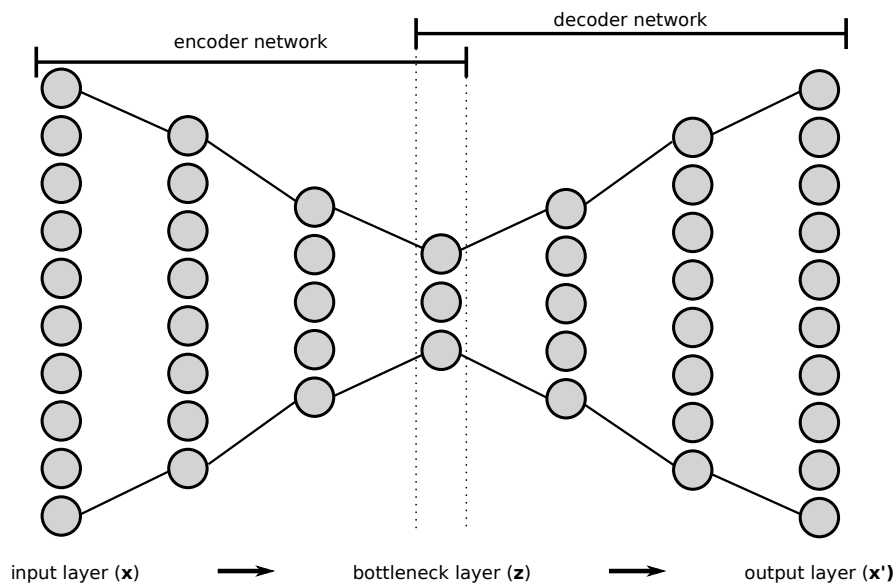


Figure 2.7: A simple Auto-Encoder. In this example, the input and output layers consist of 10 neurons, while the bottleneck layer consists of 3 neurons. To avoid clutter in the diagram, we have omitted drawing all of the connections. In a MLP Auto-Encoder, all of the layers are fully connected. In a Convolutional Auto-Encoder, the layers are arranged into grids as we describe in subsection 2.3.15: *Convolutional Neural Networks (CNN)*

$$E : \mathbf{x} \mapsto \mathbf{z} \quad (2.8)$$

$$D : \mathbf{z} \mapsto \mathbf{x}' \quad (2.9)$$

$$\mathcal{L} = (\mathbf{x} - \mathbf{x}')^2 \quad (2.10)$$

There are currently no clear rules as to how to determine the optimal size of the bottleneck layer \mathbf{z} . The dimensionality of \mathbf{z} is a new hyperparameter that we must find as we described previously in subsection 2.3.13: *Hyperparameter search*. As the regularity and similarities amongst the dataset increases, for example, in the case of a dataset consisting of only human faces, perfectly cropped and aligned in exactly the same manner, then we can afford to *decrease* the bottleneck size. As we encounter more variety in the dataset, we may have to *increase* the bottleneck size. Otherwise, we may find that the expressive power of the network is insufficient to successfully represent the full diversity of the dataset.

2.3.17 Variational Auto-Encoders (VAE)

Given suitable architecture, hyperparameters and training data, an Auto-Encoder may learn efficient representations in its bottleneck \mathbf{z} . However, in a typical Auto-Encoder we have no control over how this space is distributed. The encoder may learn a mapping such that the training examples are located in distant remote corners of the space with vast ‘empty’, unused stretches in between. In other words, an Auto-Encoder does not learn the distribution of the data and it is not a generative model. As a result of this, we are not able to generate new

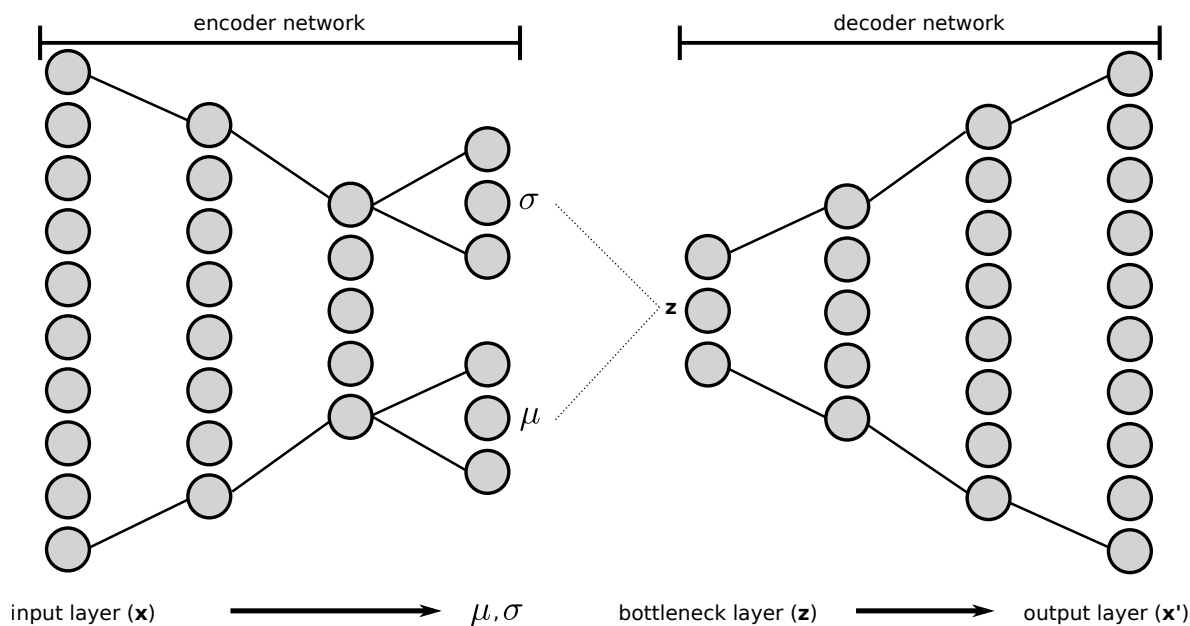


Figure 2.8: Variational Auto-Encoder. The output of the encoder network is the mean μ and standard deviation σ of a Normal distribution, from which a latent vector z is sampled $z \sim \mathcal{N}(\mu, \sigma)$. This is then fed into the decoder network for decoding.

samples that resemble the training examples. Furthermore, having compressed a particular input x to a more concise representation z , we cannot manipulate z in any meaningful way to obtain outputs that have been meaningfully modified, as we discuss in subsection 2.1.3: *Latent manipulations*.

A **Variational Auto-Encoder** (VAE) addresses this issue by enforcing structure on the bottleneck via a **regulariser**. This regulariser ensures that the bottleneck converges towards a desired distribution (D. P. Kingma & Welling, 2013), which we typically select as a multivariate standard normal distribution $\mathcal{N}(0, 1)$. In this case, the encoder does not output a z vector directly. Instead, it outputs a pair of mean μ , and standard deviation σ vectors, with the same dimensions as z . Sampling from this Gaussian distribution, produces a z vector that we can feed into the decoder network, to produce the final output. This is known as the **reparametrisation trick** of VAEs, and is illustrated in Fig. 2.8.

To ensure that the encoder's outputs μ and σ do converge to $\mathcal{N}(0, 1)$, we add to the loss function the *KL divergence* between the encoder's outputs and $\mathcal{N}(0, 1)$. In other words, we write the loss as a sum of two components. The first component is the **reconstruction loss**. This serves the same purpose as the loss in a standard Auto-Encoder in that it enforces the network to successfully reconstruct its inputs. The second component is the **latent loss**. This is the KL divergence which enforces the latent distribution, as output by the encoder, to converge to $\mathcal{N}(0, 1)$. We can formulate this as

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (\mathcal{L}_{reconstruction}^{(n)} + \mathcal{L}_{latent}^{(n)}) \quad (2.11)$$

$$\mathcal{L}_{reconstruction}^{(n)} = \sum_{k=1}^K (x_k^{(n)} - x'_k)^2 \quad (2.12)$$

$$\mathcal{L}_{latent}^{(n)} = \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^{(n)})^2 - (\mu_j^{(n)})^2 - (\sigma_j^{(n)})^2) \quad (2.13)$$

$$\mathbf{z}^{(n)} = \mu^{(n)} + \sigma^{(n)} \odot \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 1) \quad (2.14)$$

where the superscript (n) denotes the n th training example, N is the number of training examples in the given iteration, the subscript k denotes the k th element of the current data point, K is the dimensionality of the input, J is the dimensionality of latent vector \mathbf{z} , μ and σ denote the outputs of the encoder, and \odot is an element-wise multiplication,

Both the encoder and the decoder networks, are now probabilistic models. The encoder models the probability of $P(\mathbf{z} | \mathbf{x})$, while the decoder models $P(\mathbf{x} | \mathbf{z})$. Since the model has been trained enforcing a $\mathcal{N}(0, 1)$ latent distribution, we can sample a random $\mathbf{z} \sim \mathcal{N}(0, 1)$, and run this random \mathbf{z} through the decoder to generate new data consistent with the training examples. In other words, the VAE's decoder is a *generative model*, and we can benefit from the types of meaningful control that we discuss in subsection 2.1.3: *Latent manipulations*.

We use a VAE as the basis for our research in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, where we present a software tool that can train in realtime on a live video feed while allowing for the realtime manipulation of a number of hyperparameters.

We noticed that the performance of the VAE was highly dependent on the dimensions of the inputs and the bottleneck. We observed that in certain cases the VAE was able to reconstruct its inputs very accurately, but the decoder was not able to generate samples that were consistent with the training examples when decoding a random $\mathbf{z} \sim \mathcal{N}(0, 1)$. This implies that the latent distribution does not converge to $\mathcal{N}(0, 1)$. In other cases we saw that decoding random $\mathbf{z} \sim \mathcal{N}(0, 1)$ did produce very consistent samples, however, the VAE was unable to reconstruct its inputs accurately.

We found that this is due to an imbalance between the reconstruction loss and latent loss in eqn. (2.11), and at the time of our research, this was not mentioned in any literature, or the many open-source examples found on the internet. We implemented a solution whereby we normalise both the reconstruction loss and latent loss, dividing by the dimensionality of input and bottleneck layers respectively. We then *weight* the latent loss by a new hyperparameter W_{KL} , which we include in our hyperparameter search. Our new formulation can be seen below.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (\mathcal{L}_{reconstruction}^{(n)} + W_{KL} \mathcal{L}_{latent}^{(n)}) \quad (2.15)$$

$$\mathcal{L}_{reconstruction}^{(n)} = \frac{1}{K} \sum_{k=1}^K (x_k^{(n)} - x'_k)^2 \quad (2.16)$$

$$\mathcal{L}_{latent}^{(n)} = \frac{1}{J} \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^{(n)})^2 - (\mu_j^{(n)})^2 - (\sigma_j^{(n)})^2) \quad (2.17)$$

We found via a thorough hyperparameter search, that a W_{KL} value in the range of 0.01–0.1 provides the most desirable results, depending on our priority. A higher value towards 0.1, provides higher quality random samples, while slightly compromising reconstruction quality. On the other hand, lower values towards 0.01, provides higher quality reconstructions, while slightly compromising random sample quality. Outside of this range, we found that the compromise was too large to be useful. But within this range, the overall quality was generally acceptable. Since this weighted sum is performed on normalised loss components, W_{KL} is independent of input or bottleneck dimensions. We tested this on images, with many different datasets, assessing the reconstruction and random sample qualities visually.

This was also later examined independently by Higgins et al in their *Beta-VAE* (Higgins et al., 2017). The researchers propose a similar solution to ours, performing a weighted sum of the loss components. However, Higgins et al do not normalise the loss components before summing. They simply weight the latent loss by a hyperparameter β . While β does allow shifting balance between reconstruction quality vs random sample quality, it is still highly dependent on the input and bottleneck dimensions. We find this to be problematic, at least for our use case. If for example, we change the bottleneck dimensions, and the reconstructions comes out more blurry. We cannot be sure if this is directly a result of the bottleneck dimensions changing, and affecting the expressive power of the network, or because the balance of the latent loss to reconstruction loss has shifted dramatically. In other words, having found an optimal β value through trial and error, if we were to change the input or bottleneck dimensions, our optimal β would no longer perform in the same manner, and would not be optimal anymore. We would either have to find a new value through trial and error, or calculate by hand what the new optimal β should be, based on the old dimensions and new dimensions. For this reason, we prefer to weight the normalised loss components, by our W_{KL} hyperparameter.

We discuss these experiments and our findings in more detail in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*.

2.3.18 Deep Convolutional Generative Adversarial Networks (DCGAN)

It's worth noting, that using L2 or L1 pixel-wise loss to compare images — for example, to compare the input image \mathbf{x} and output image \mathbf{x}' of a VAE — results in the model producing blurry images (I. Goodfellow et al., 2014; Theis et al., 2016; Larsen et al., 2016; Dumoulin et al., 2017). This is a known issue with using these loss functions on a per-element basis in raw data-space, and is in fact not limited to images. In the case of our own work with VAEs,

we address this by implementing *Multi-Scale Structural Similarity Index Measure* (MS-SSIM) (Z. Wang et al., 2004) as a reconstruction loss. We discuss this in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, and show that the results are far superior to using L1 or L2 pixel-wise loss. However, MS-SSIM is a metric that has been specifically designed to compare the similarity of *images*, and it cannot be used in other domains.

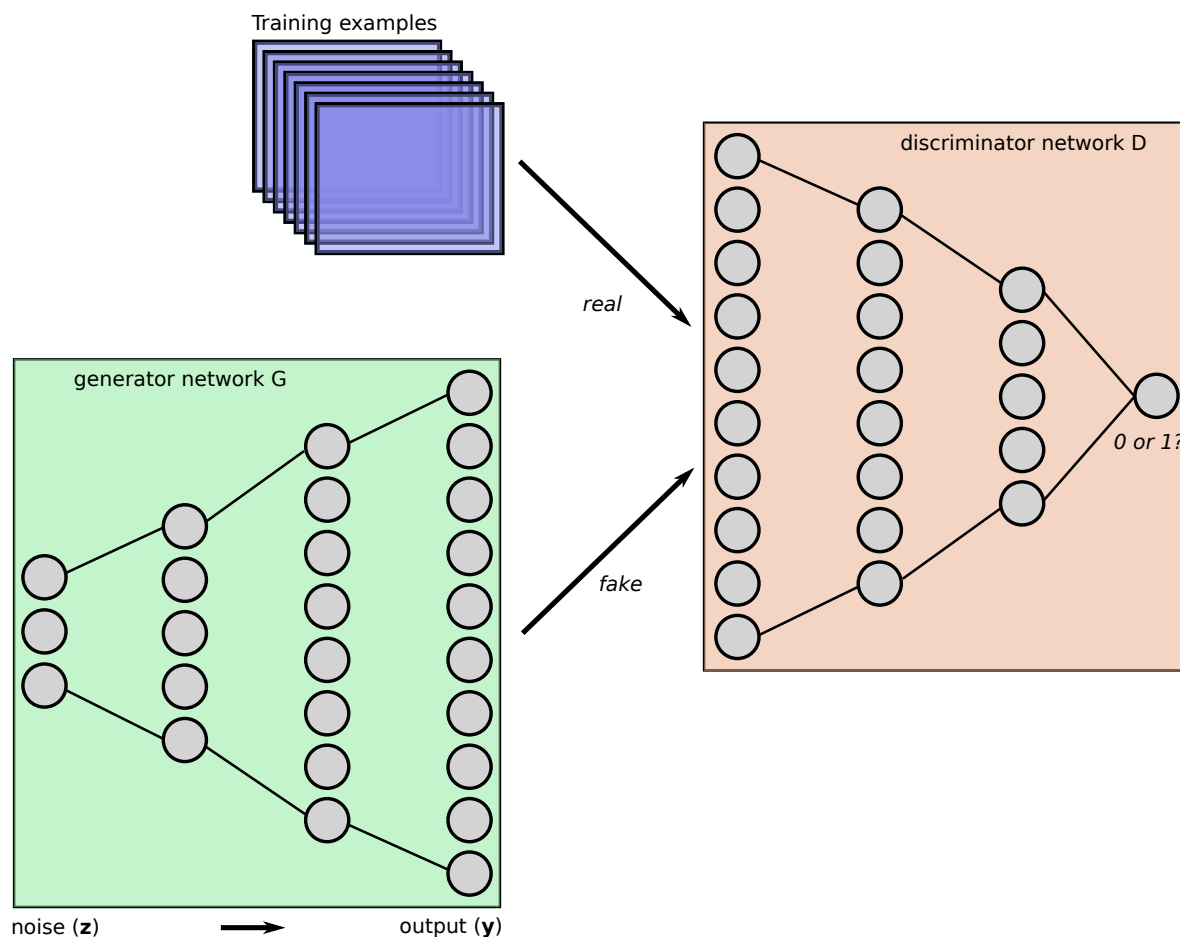


Figure 2.9: Generative Adversarial Network

Generative Adversarial Networks (GAN) were introduced to address this issue on a more generic and theoretical level (I. Goodfellow et al., 2014). The motivation behind a GAN, is that instead of *explicitly formulating* a loss function to measure the similarity between data points — for example comparing a *generated* image to a *real* image, we introduce a second Deep Neural Network to *learn* such a metric. We construct a **Generator network** G which captures the distribution of the data, in that it learns to map a random noise vector $\mathbf{z} \sim \mathcal{N}(0, 1)$ to some output \mathbf{y} . We also construct a **Discriminator Network** D , which learns to distinguish between **real** samples — i.e. sampled from the training data — and **fake** — i.e. produced by the Generator network. Since we are primarily interested in images in this thesis, given a noise vector \mathbf{z} , $G(\mathbf{z})$ will produce a *fake* image. And given some image \mathbf{x} , $D(\mathbf{x})$ will output a single scalar that represents the probability that \mathbf{x} is *real*, i.e. taken from the training data. A high level schematic of this can be seen in Fig. 2.9.

In this case, the loss function of a GAN constitutes two parts. The Discriminator network is trained to guess whether a given sample is real or fake, in other words, to *maximise* the probability of correctly assigning the real or fake label to the samples provided. The Generator network is trained to generate samples which can fool the Discriminator, in other words, to *minimise* $\log(1 - D(G(\mathbf{z})))$. For this reason, the optimisation of a GAN is no longer a *minimisation* problem. Rather, it involves finding a *Nash equilibrium* of a non-convex minimax game with a value function $V(G, D)$ given by (I. Goodfellow et al., 2014)

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (2.18)$$

Just as we can include convolutional layers in VAEs, we can also include convolutional layers in the Generators and Discriminators of GANs. These are then called **Deep Convolutional Generative Adversarial Networks** (DCGAN). In this thesis, as we are focusing on pixel-based raster images, we are generally interested in architectures with 2D convolutions, and such 2D DCGANs have seen great progress in terms of resolution and quality in recent years, expanding from 64x64 pixels with many deformities (Radford et al., 2015), to 1024x1024 pixels with almost no undesirable artefacts (Karras et al., 2017, 2019, 2020). While we do not work with them in this thesis, GANs have also been used to model 3D voxel geometry (J. Wu et al., 2016; C. Lin et al., 2016; Gwak et al., 2017; Liu et al., 2017), and audio (Engel et al., 2019).

While GANs do demonstrate the ability to produce samples consistent with the training data, they can also be very difficult to train and suffer from a number of issues (Radford et al., 2015; Chen et al., 2016; Salimans et al., 2016; Berthelot et al., 2017; Arjovsky et al., 2017). For this reason, a vast amount of research has been conducted in recent years trying to improve the stability and quality of GAN training. As a result, many variations have been proposed. Colloquially known as *The GAN Zoo*, at the time of writing, there are over 500 GAN variations (Hindupur, n.d.)³⁰. Our research is not necessarily concerned with internal or training improvements to GANS per se, so we will not dwell on these variations.

Our research is concerned at a very high level with two broad categories of GANs. These reflect the classes of generative models that we discuss in subsection 2.1.2: *Conditional generative models* and subsection 2.1.1: *Unconditional generative models*.

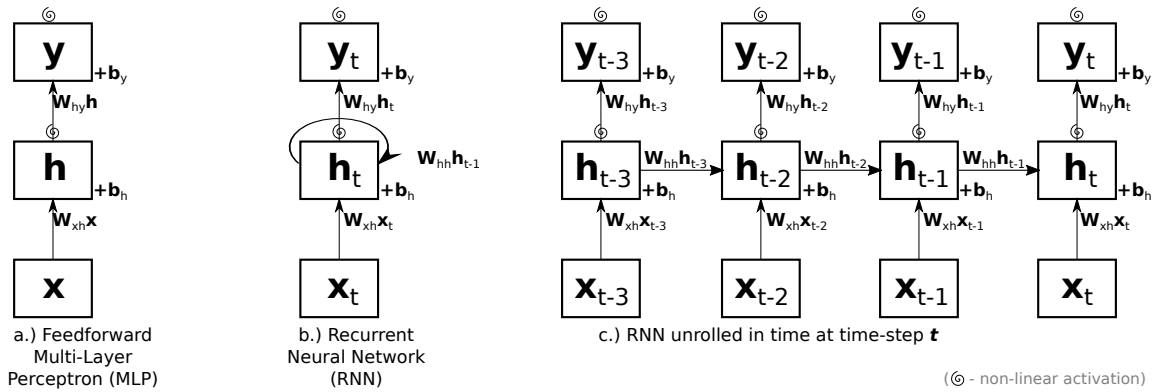
In chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we investigate *conditional* DCGANs. In particular, we primarily work with *pix2pix* (Isola et al., 2016), although we also test our method with *pix2pixHD* (T. C. Wang et al., 2018).

In chapter 6: *Deep Meditations: Latent storytelling*, we investigate *unconditional* DCGANs. In particular, we primarily work with ProGAN (Karras et al., 2017), although we also test our method with BigGAN (Brock et al., 2019) and StyleGAN (Karras et al., 2019).

2.3.19 Recurrent Neural Networks (RNN)

One of the early studies that we present in section 3.3: *Realtime interactive text generation with RNN ensembles*, is an interactive character-level sequence model. For this we use a **Long**

³⁰These include applications to different domains.



$\mathbf{h} = f(\mathbf{x})$ where

$$f(\mathbf{x}) = \varphi_h(\mathbf{W}_{xh}\mathbf{x} + \mathbf{b}_h)$$

$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$ where

$$f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \varphi_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

Figure 2.10: MLP and RNN with single hidden layer.

Short-Term Memory (LSTM) Recurrent Neural Network (RNN). In this section, we provide the necessary technical background for this architecture.

As we have briefly mentioned in subsection 2.3.4: *Feed-forward (FNN) vs Recurrent Neural Networks (RNN)*, RNNs are ANNs with recurrent connections. While there are many different variations (Graves, 2008), a very simple RNN can be thought of as a MLP with recurrent connections on one or more layers. These recurrent connections carry information forward from previous timesteps, and allow the recurrent neurons to maintain an *internal state*. This enables RNNs to create and process *memories* from past inputs, learn temporal regularities and model non-linear dynamical systems (Graves, 2008; Sutskever, 2013). With a richer internal state and greater computational capacity than other sequence models such as Hidden Markov Models, RNNs can be seen as *general purpose computers* that can learn *programs* (Schmidhuber, 2012b; Graves et al., 2014; Schmidhuber, 2015).

More formally, the value of a recurrent layer at a particular timestep is also dependent on the *state from the previous timestep*. For a RNN with a single hidden layer as shown in Fig. 2.10, this can be formulated by augmenting eqn. (2.3) with

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.19}$$

$$\text{where } f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \varphi_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h), \tag{2.20}$$

and \mathbf{x}_t and \mathbf{h}_t are respectively the *input vector* and *hidden state vector* at time t . Note that \mathbf{W}_{hh} is the *recurrent weights matrix*, i.e. the influence of the previous timestep's hidden state on the current timestep's hidden state.

This architecture can be expanded to multi-layer, or *stacked RNNs*, by introducing additional weight matrices and bias vectors for inter-layer and inter-time relationships, as well as *skip-connections* between layers or forward looking *bi-directional* connections, to capture richer time-dependent behaviours (Schmidhuber, 1992; Schuster & Paliwal, 1997).

Backpropagation Through Time

Training RNNs is still performed as we described in subsection 2.3.10: *Gradient descent and backpropagation*. However, since RNNs also have a time component, we also have to perform **backpropagation through time (BPTT)** (Werbos, 1990; Sutskever, 2013). As the name implies, the RNN is unrolled in time for every step of the sequence as shown in Fig. 2.10. It can then be trained with backpropagation like a regular deep network, with the constraint that weights are shared across timesteps. Thus RNNs can be thought of as *deep in time* as well as *deep in layers*.

However, for very long sequences this will result in very deep networks, which can be rather slow and inefficient to train. One way to optimise this is to segment the full sequence of length L timesteps into shorter segments of length l timesteps. We can then run BPTT on each segment, and train the network much quicker, but at the cost of losing long-term temporal relationships greater than l timesteps. For very long sequences where these long-term temporal relationships are required, an efficient approximation is **Truncated Backpropagation Through Time** (Sutskever, 2013). With this method we run BPTT on each segment *sequentially*, without shuffling the order of the segments and without resetting the RNN's internal state between segments. So even though the gradients are only propagated back l timesteps, the network has the potential to remember further back in time via its internal state.

Objective

Given a training set S of input-target pairs $(\mathbf{x} \in X, \hat{\mathbf{y}} \in \hat{Y})$, and Θ the parameters of our network, our training objective is to find the set of parameters Θ_{ML} with the *maximum likelihood* (ML), i.e. that maximises the probability of training set S with (Graves, 2008)

$$\Theta_{ML} = \arg \max_{\theta} P(S | \Theta) \tag{2.21}$$

$$= \arg \max_{\theta} \prod_{(\mathbf{x}, \hat{\mathbf{y}})}^S P(\hat{\mathbf{y}} | \mathbf{x}, \Theta). \tag{2.22}$$

Since the logarithm is a monotonic function, a common method for maximizing its likelihood is minimizing its negative logarithm, also known as the **Negative Log Likelihood (NLL)**, **Hamiltonian** or **surprisal** (H. W. Lin & Tegmark, 2016). We can then define our objective function \mathcal{L} as

$$\mathcal{L} = -\ln \prod_{(\mathbf{x}, \hat{\mathbf{y}})}^S P(\hat{\mathbf{y}} | \mathbf{x}, \Theta) \tag{2.23}$$

$$= -\sum_{(\mathbf{x}, \hat{\mathbf{y}})}^S \ln P(\hat{\mathbf{y}} | \mathbf{x}, \Theta). \tag{2.24}$$

Long Short-Term Memory (LSTM)

While BPTT and variants can solve the *performance* issue of training RNNs deep in time (i.e. on long sequences), backpropagating error gradients is still likely to fail due to the *vanishing and exploding gradients* problem, whereby the backpropagated error can result in gradients exponentially diminishing to zero or exploding towards infinity (Hochreiter, 1991; Bengio et al., 1994). This has made it very difficult to use RNNs on real-world problems with long sequences. In order to address this problem, Hochreiter and Schmidhuber (1997) have proposed an architecture for recurrent units known as **Long Short-Term Memory** (LSTM). An LSTM *cell* is a recurrent unit which — in addition to the normal *hidden state* of a recurrent unit — contains a **memory cell**. Access to the memory cells are controlled via *gates* which open and close as desired, allowing the information in the cell to be preserved for extended periods of time. Gates are implemented as logistic sigmoid functions, thus their state (*open* vs *closed*) is continuous and differentiable. This allows their behaviour to be learnt during training along with the network weights and biases, with gradient descent and backpropagation.

Initially, Hochreiter and Schmidhuber (1997) introduced *input* and *output* gates to control writing to and reading from the memory cells. Subsequently, Gers et al. (2000) proposed *forget* gates, allowing LSTM cells to learn when to reset themselves. With this architecture, LSTM networks can preserve and process information from many timesteps in the past.

Similar to the simple RNN demonstrated above, at timestep t , an LSTM cell outputs a *hidden state vector* \mathbf{h}_t to the rest of the network. But in addition to this, it also contains a *memory cell* \mathbf{c}_t of the same dimensions. Collectively these two vectors constitute the *internal state* of the cell and replace eqn. (2.19) with

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (2.25)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c), \quad (2.26)$$

where \odot is an element-wise multiplication and \mathbf{i}_t , \mathbf{o}_t , \mathbf{f}_t are respectively input, output and forget gates at time t . The gates have the same dimensions as \mathbf{h}_t and are defined as

$$\mathbf{i}_t = \varphi(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) \quad (2.27)$$

$$\mathbf{o}_t = \varphi(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) \quad (2.28)$$

$$\mathbf{f}_t = \varphi(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f), \quad (2.29)$$

where φ denotes the logistic sigmoid function. Both φ and \tanh are applied element-wise. While the LSTM architecture is generally successful in overcoming the vanishing gradients problem, gradients can still tend to become quite large. This can be prevented by clipping the gradients. A simple method is to either clip them element-wise (potentially changing the direction of the gradient) or clip the L2 norm by re-scaling the elements (Pascanu et al., 2012). A computationally more expensive, but more stable solution was proposed by Pascanu et al. (2013), in which gradients are rescaled according to the global norm.

LSTM Variants

In recent years, many variations on the LSTM architecture has been introduced. These include **Peephole connections** (Gers & Schmidhuber, 2000), in which the gates also learn to respond to the cells' current value (from previous timestep for input and forget gates, adding a $\mathbf{W}_c \mathbf{c}_{t-1}$ term; and current timestep for output gate, adding a $\mathbf{W}_c \mathbf{c}_t$ term). This has been shown to enable the LSTM to learn more precise timing and rhythmic information. In addition, simplifications have been proposed by pairing and omitting (effectively merging) gates, such as *No Input Gate* and *No Output Gate* LSTM cells (Z. Wu & King, 2016). An even simpler architecture was proposed in Cho et al. (2014) which introduces **Gated Recurrent Units** (GRU). These also have only two gates — *reset* and *update*, but further simplify the LSTM architecture by omitting the separate memory cell and storing the long term memory in the hidden state vector.

Due to fewer gates — and thus fewer weights and biases — these variants are computationally less expensive. However, ignoring computational requirements, they have been shown to perform roughly the same as standard three-gate LSTMs, occasionally performing marginally better on some tasks, and occasionally worse on others depending on the domain (Greff et al., 2015; Jozefowicz et al., 2015; Nayebi & Vitelli, 2015).

Sequence generation with Recurrent Neural Networks

Combining these techniques, and with increased compute power, large training sets and many new insights into training deep RNNs (Graves, 2012; Sutskever, 2013; Zaremba et al., 2014; Pham et al., 2014; Karpathy et al., 2015), LSTMs and its variants are having lots of success not only in sequence *classification* (Graves et al., 2009; Hinton et al., 2012; Pham et al., 2014), but also in sequence *generation* in domains such as music (Eck & Schmidhuber, 2002; Boulanger-Lewandowski et al., 2012; Nayebi & Vitelli, 2015; Sturm, 2015), text (Sutskever et al., 2011; Sutskever, 2013), handwriting (Graves, 2013), images (Gregor et al., 2015), machine translation (Sutskever et al., 2014), Chinese characters (Nayebi & Vitelli, 2015; Ha, 2015), speech synthesis (Z. Wu & King, 2016) and even choreography (Crnkovic-friis & Crnkovic-friis, 2016).

2.3.20 Monte Carlo Tree Search

In another study that we present in section 3.2: *Collaborative generative sketching with MCTS and CNNs*, we use an agent-based approach to collaborative, generative sketching. The agent-based method that we use is *Monte Carlo Tree Search* (MCTS). This is not a DNN architecture per se. However, it is an agent-based approach to ML, and is used for example by DeepMind in their AlphaGo (Silver et al., 2016) software, that beat world Go champions Lee Sedol and Fan Hui; and AlphaZero (Silver et al., 2018) software, that beat human champions and other programs at the games of Go, Chess and Shogi.

MCTS is a very efficient, probabilistic approach to exploring a vast decision space. A detailed explanation and survey of MCTS can be found in (Browne et al., 2012), but for completeness we will include a summary below.

We represent our problem as a discrete-time *Markov Decision Process* (MDP) (Bellman, 1957). At a particular timestep, the system is in a *state* s . The agent can take an *action* a (out

of a number of available actions n_a) to go to a new state s' , and will be given a *reward* $R_a(s, s')$ for doing so.

Instead of trying to simulate *every* possible action at every timestep, MCTS uses a heuristic to explore only what might be the most *promising* branches of the decision tree. At every timestep, many simulations are run, but only from carefully chosen nodes of the decision tree. The algorithm decides which node to expand and run a simulation from based on a balance between *exploiting* known high rewards for nodes already expanded, and *exploring* new nodes, not yet expanded, with unknown rewards. Many rollouts (i.e. simulations) are performed at each timestep, and the simulated rewards are backpropagated along the partially expanded decision tree, and accumulated in each node, assigning an estimated *value* to each potential action.

The *tree policy* decides which action (i.e. child node) to select and unroll (i.e. run a simulation from), and is given by the UCT formula

$$\frac{v}{n_c} + k\sqrt{\frac{2 \ln(n_t)}{n_c}} \quad (2.30)$$

where v is the value of the child node, n_c is the number of times the child node has been visited, k is the exploration parameter (theoretically equal to $\sqrt{2}$ but usually chosen empirically) and n_t is the total number of simulations for the node considered. The *default policy* decides how to choose actions during the rollout (i.e. simulation). We use a *random rollout*, i.e. actions are selected randomly from the n_a actions.

In a typical two-player zero-sum game such as tic-tac-toe, MCTS simulations will run until they reach an *end game state*, where a simple reward of 1 is backpropagated if the agent wins, -1 is backpropagated if the agent loses, and 0 is backpropagated if the agent draws. In situations where rollouts could potentially last for many timesteps, combinatorially increasing the size of the decision tree and making it very inefficient to run the simulation upto an end state such as in the game of Go (Silver et al., 2016), it is common to end the simulation after a predefined *maximum rollout depth*. In this case another heuristic is used to evaluate the end-state of the rollout, and that is backpropagated as the reward. This process is repeated, nodes are expanded and simulations run usually until a predefined time budget (i.e. maximum milliseconds per timestep) is reached, after which a final decision is made by choosing the action which was visited the most (some implementations use different policies to choose the final action, examples can be found in (Browne et al., 2012)).

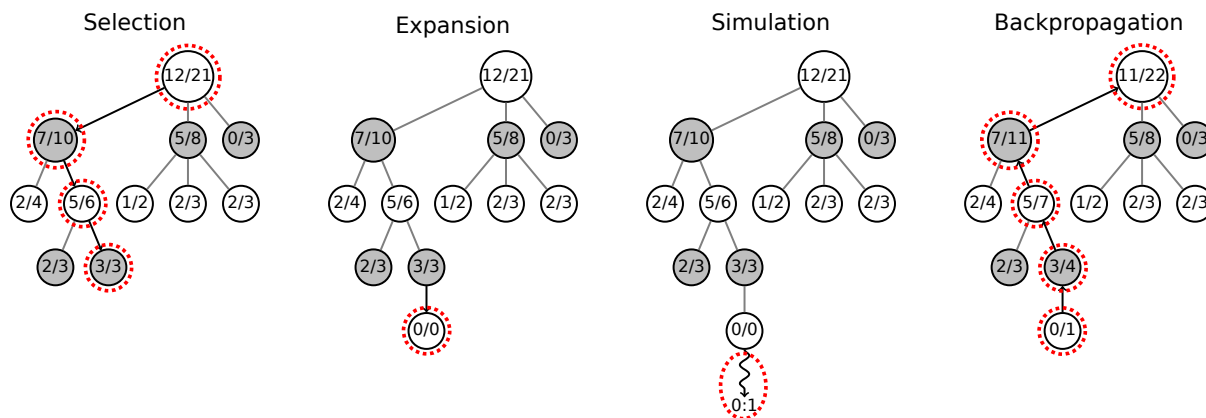


Figure 2.11: Overview of the steps of MCTS

2.4 Conclusion

In this section, we presented key concepts and the technical foundations that are required for the remainder of this thesis. We discussed *discriminative* and *generative* models, within generative models we discussed *conditional* and *unconditional* generative models, and we presented the deep architectures that we work with in this thesis. These include Recurrent Neural Networks, Convolutional Neural Networks, Variational Auto Encoders and Generative Adversarial Networks. In the following chapters, we will present a number of different approaches that we followed using these different architectures, to build *Deep Visual Instruments*, and we will discuss the affordances and limitations of each approach.

Chapter 3

Realtime sequence generation with continuous control

3.1 Introduction

In the previous chapters, we presented a number of examples and applications of Deep Neural Networks being used for creative media generation across a wide range of media. While these examples have often demonstrated very impressive results, the topic of Meaningful Human Control¹, and in particular Realtime Continuous Control², remains an open issue.

One of the challenges that we wish to overcome, is that in many of these existing systems the entire output is often generated in a single timestep. For example, when working with a generative *image* model³ such as ProGAN, we can sample a random $z \sim \mathcal{N}(0, 1)$, and an entire 1024x1024 pixel image is generated in one go. We do have the ability to modify this image via latent manipulations⁴, however we envisage a system which allows more control *while the image is being generated*. In other words, we imagine an image being sketched out by a software in realtime — as if we are watching a person draw the image — and we can somehow guide the software while it is sketching. Another example concerns generative *text* models such as *Char-RNN*⁵. These have become very popular, as they can produce believable, though often hilariously erroneous, pieces of text. However, at the time of our research, there are no tools available that allows a human to have any kind of meaningful control over the text generated, let alone in a Realtime Continuous manner.

In other words, we are interested in developing generative systems which *continually* and *incrementally* produce some kind of an output — for example drawing a sketch, or typing out text. And at any time, a person can meaningfully interact with and guide the system with Realtime Continuous Control. As we described in section 1.4: *Visual instruments: Realtime Continuous Control*, we use musical instruments as a metaphor to think of these systems as *visual instruments*. These are not *used* by a *user*, but are *played* by a *performer*.

¹section 1.3: *Meaningful Human Control*

²section 1.4: *Visual instruments: Realtime Continuous Control*

³section 2.1: *Generative models*

⁴subsection 2.1.3: *Latent manipulations*

⁵section 2.2.7: *Char-RNN*

In this chapter, we present two studies that investigate this topic, and we specifically focus on the two examples that we mention above: drawing sketches, and generating text.

3.2 Collaborative generative sketching with MCTS and CNNs

3.2.1 Introduction

In this section, we present a method for collaborative, generative sketching using *Monte Carlo Tree Search* (MCTS) and *discriminative Convolutional Neural Networks* (CNN).

Even though this experiment was not able to produce the realistic results that we were hoping for, based on our findings and the questions that our research set out to address, our paper (Akten & Grierson, 2016a) was accepted to the *Constructive Machine Learning Workshop* at the *Thirtieth Conference on Neural Information Processing Systems* (NeurIPS) in 2016. Furthermore, the system that we described and were hoping to build, was later replicated — though with a very different technical approach — in Google’s *Sketch-RNN* (Ha & Eck, 2017)⁶ and the associated online collaborative drawing applications⁷.

3.2.2 Background

As we mention above, in a typical Deep Learning based creative workflow, the entire output is generated in a single timestep. This is especially true of images. One of the main reasons for this, is that images are represented as bitmaps, a dense grid of *(Red, Green, Blue)* tuples. Generative models such as VAEs⁸ or GANs⁹ model this representation, so that when we take a sample from the latent distribution of such a model, this produces an entire image in a single timestep.

There are a few notable exceptions to this. Auto-regressive models such as *PixelRNN* (van den Oord, Kalchbrenner, & Kavukcuoglu, 2016) and *PixelCNN* (van den Oord, Kalchbrenner, Vinyals, et al., 2016) also operate on bitmap images. However, in these models, each pixel value is conditioned on all of the previously generated pixel values. Generating an image with PixelCNN or PixelRNN does not happen in a single timestep. Instead it happens pixel by pixel. Technically this could allow for Realtime Continuous Control, if a suitable interface were designed. However, generating an image pixel by pixel in this way is not only incredibly slow, we do not believe that a bitmap representation is optimal for Meaningful Human Control in this context. The manipulations that we seek to make possible, are not on a *pixel* level, but are at a higher, semantic level.

Another exception is Google’s *Sketch-RNN*. Sketch-RNN was released a year after we presented our research first at an internal workshop at Google’s offices in San Francisco, and later at the *Constructive Machine Learning Workshop* at NeurIPS 2016. The researchers use a very different technical implementation to what we describe below, however their motivations and the functionality of the system and web applications that they develop is the same as what we propose below and in our original paper. This is to say that they develop a system that is able to draw arbitrary sketches of various different categories. Furthermore, a user can initiate the sketch, and then the system can recognise what the user has drawn — for example, the tail of a cat — and then complete the rest of the drawing accordingly. In Sketch-RNN, the authors train

⁶section 2.2.7: *Sketch-RNN (2017)*

⁷<https://magenta.tensorflow.org/sketch-rnn-demo>

⁸subsection 2.3.17: *Variational Auto-Encoders (VAE)*

⁹subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*

a generative sequence model on *vector representations* of strokes. One of the biggest challenges of such an approach, is having a large enough dataset of *vector drawings* to train on. At the time of our research, there were no such publicly available labelled datasets. For Sketch-RNN however, the researchers were able to use an internal dataset of over 50 million vector drawings, which was made publicly available later that year (*Quick, Draw! Dataset*, 2017).

3.2.3 Overview

In this study, we aim to develop an agent that can wander around the screen, trying to sketch out drawings that resemble specific objects as chosen by a person. The agent doesn't create a drawing that mimics a *specific target picture*. That is a relatively well established area of research and artistic inquiry (Tresset & Deussen, 2014; Smith & Leymarie, 2017). Instead, in this study, our aim is to allow the agent to take any actions it chooses, and follow any path that resembles a desired *image class*. For example, if the agent is asked to draw a *cat*, the agent will sketch *any* cat that it imagines, not a specific cat presented to it in any picture.

Most crucially, the agent should not decide the entirety of what the final image would be in an instant. I.e. it should not sample a random cat image from a latent space (as one might do with a GAN or VAE), and then sketch what it has sampled. The final image should not be decided the instant the agent starts drawing. The agent should know what a cat looks like, it should plan ahead and start drawing, but continuously adapt its actions as it responds to its own drawings. This would allow for a person to interact with the agent while it is drawing. A user could push the agent around (e.g. by exerting forces on it) and directly affect the drawing. In addition, the user could directly draw on the canvas too. Seeing this, the agent should adapt. E.g. while the agent is drawing, if the human draws a partial sketch that resembles a tail, the agent might recognize that, and draw the rest of the cat to fit to the tail drawn by the human. The agent should continuously monitor the page, and intelligently incorporate the current state of the drawing into its future plans. It should see, imagine, plan and respond. This should be a collaborative process, that evolves in realtime, continuously and interactively.

There has been prior work in generating *bitmap images* based on image classes, and prior work in generating *drawings* that *mimic specific target bitmap images*. However, to our knowledge there has not been any prior work in directly generating vector drawings based on image classes, let alone in a way that is able to adapt and collaborate based on input from a person. Google's Sketch-RNN, which was released the year after this study, also does exactly this.

As a guiding framework, our system is designed around these basic principles:

- **Creativity:** as an efficient method of searching a large space of actions or decisions,
- **Imagination:** as the ability to simulate many different scenarios and outcomes,
- **Evaluation:** as the ability to evaluate both the current situation, as well as the outcome of many imagined actions and scenarios, against some desired criterion,
- **Collaboration:** as the ability to respond — somewhat intelligently and creatively — in realtime, to some kind of external input, such as the actions of another user.

3.2.4 System description

Our agent acts similar to a LOGO Turtle (Papert et al., 1980). It can move forward, and rotate left or right by a number of degrees. It can move leaving a trail (analogous to *pendown* in Turtle) or without leaving a trail (analogous to *penup* in Turtle).

At the heart of the decision making process driving the agent, is *Monte Carlo Tree Search* (MCTS) (Browne et al., 2012). MCTS is a very efficient, probabilistic approach to exploring a vast decision space while aiming to maximise future cumulative rewards (Browne et al., 2012). We present a more detailed explanation of MCTS in the previous chapter subsection 2.3.20: *Monte Carlo Tree Search*.

Our system runs at approximately 2-5 frames per second on a medium level laptop, depending on various parameters such as *rollout depth*. At every timestep, the following actions are performed (for a diagrammatic overview, please see Fig. 3.1):

1. MCTS performs many simulations, known as *rollouts*, where the agent takes random actions. These rollouts are performed for an arbitrary number of timesteps into the future. We consider this to be the agent *imagining* what would happen if it took particular actions.
2. At the end of each rollout, the simulated trajectory or *drawing*, is rendered into a texture and fed into an image classifier, such as a CNN trained for image classification. This is to *evaluate* how much the simulated, imagined drawing resembles the desired image *class*.
3. The desired *class probability* returned from the classifier is backpropagated through the partially expanded decision tree as a *reward* towards choosing the optimal action for that timestep.
4. Hundreds or even thousands of simulations are performed, trajectories imagined and evaluated per timestep. Because of the probabilistic nature of MCTS, and the balance between *exploration vs exploitation*, the system converges towards imagining actions and trajectories which are more likely to produce desired outcomes — i.e. drawings with higher resemblance to the desired image class.
5. When a predetermined time budget is reached (e.g. 100ms), MCTS stops simulating new actions and trajectories, and picks the most *robust child*, i.e. the most visited and thus *promising* action. (Using a predetermined time budget allows the system to remain realtime at interactive rates. Smaller time budgets allow higher framerate at the cost of less optimal actions, while higher time budgets drop the framerate but allow potentially more optimal actions).
6. The agent then performs the selected action to make that move.
7. At the next timestep, this process of *imagining and evaluating* is repeated.

In summary, at every timestep, hundreds of simulations are performed imagining hundreds of alternative actions and paths. Our system is able to *efficiently search* this very large space of actions via MCTS’s optimal branch selection; it is able to *imagine* the outcome of many different scenarios via the MCTS rollouts; it is able to *evaluate* situations — both current and

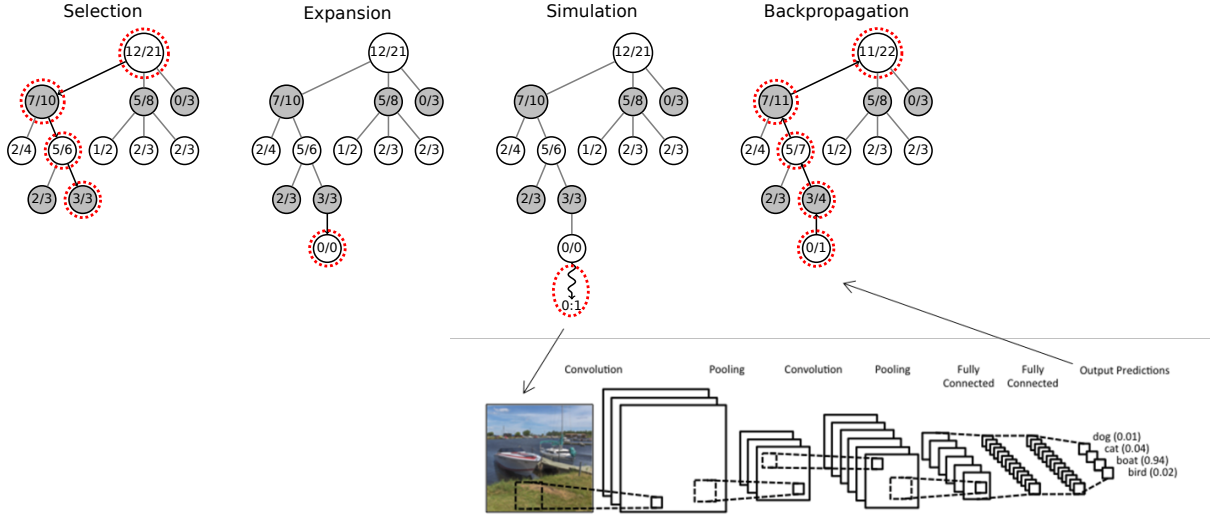


Figure 3.1: Overview of how MCTS and the CNN is integrated

imagined, also incorporating inputs from a user — via the classifier; and it is able to *integrate* all of these into its response and future plans, resulting in a *collaboration*.

Interestingly, parallel to our work, a similar architecture using MCTS to select actions where a CNN is used to evaluate imagined rollouts was also later used by Google Deepmind in their seminal AlphaGo, which beat human Go champions Lee Sedol and Fan Hui (Silver et al., 2016).

The agent can move forward with n_{speeds} number of different speeds to choose from, ranging from s_{min} to s_{max} . It can also rotate with $n_{rotations}$ number of different rotations to choose from, ranging from $-r_{max}$ to $+r_{max}$. In other words, the agent can turn anywhere from $-r_{max}$ to $+r_{max}$, in $\frac{(2 * r_{max})}{(n_{rotations} - 1)}$ degree increments. Thus the agent has $n_a = n_{speeds} * n_{rotations}$ total number of actions to choose from. We have tested a wide range of configurations of parameters, and an example configuration is as follows:

$$n_{speeds} := 2, s_{min} := 0, s_{max} := 2, r_{max} := 30, n_{rotations} := 7 \implies n_a = 14$$

For more details please refer to the paper (Akten & Grierson, 2016a)

Classifiers

We performed tests with three different classifiers:

Multinomial Logistic Regression (MLR) trained on MNIST The first model we use is a simple *Multinomial Logistic Regression* (MLR) that we train on the MNIST dataset of handwritten digits (LeCun & Cortes, 2010). This model is simply a softmax activation (Bishop, 2006) on a linear transformation of the inputs, and can be formulated as

$$\mathbf{y} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (3.1)$$

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (3.2)$$

where \mathbf{x} is the input vector containing the flattened input image pixels, \mathbf{W} and \mathbf{b} are respectively the *weights matrix* and *bias vector* to be learnt, and *softmax* is a function that converts a vector of log probabilities to a normalized probability distribution.

We train this model using stochastic gradient descent and backpropagation¹⁰ trying to minimise the cross-entropy (Bishop, 2006) between y the predicted probability distribution of our model, and y' the true distribution, i.e. the training data.

$$H(y', y) = - \sum_i y'_i \log(y_i) \quad (3.3)$$

Unsurprisingly, this model does not generalise very well, scoring approximately 90% on the validation dataset. Since the model is simply a softmax operating directly on a linear transformation of the raw pixel data, it is very restrictive on the types of inputs that it can classify, and does not provide much translational, rotational or scale invariance. In other words, as a *classifier*, it is far from optimal. This is important to note, as we discover that *less* optimal classifiers, perform *better* in a generative context. We will discuss this in more detail shortly.

LeNet (Convolutional Neural Network) trained on MNIST The second model we use, is a *Convolutional Neural Network* (CNN)¹¹ very similar to the classic LeNet5 (LeCun et al., 1989), that we also train on the MNIST dataset. We use two convolutional stacks, each stack consisting of 5x5 convolution kernels, followed by a rectified linear unit (ReLU), followed by a max-pool layer. After the two convolution stacks we use a dense fully connected layer with a softmax to convert the log probabilities into a probability distribution. We then train with stochastic gradient descent and backpropagation to minimise the cross-entropy.

To prevent over-fitting, we use dropout regularization (Srivastava et al., 2014). During training, neurons are randomly omitted with a fixed probability to prevent them from being too specialized. During inference (feed-forward), all neurons are enabled. In effect this is likened to training an *ensemble*, where many different networks with different architectures are trained simultaneously and averaged for prediction. We use a dropout probability of 50%.

This model scores 99.2% accuracy on the validation data and proves to be much more resilient to noise compared to the MLR. It demonstrates very good translation and scale invariance, with acceptable rotational invariance.

Inception-v3 (Convolutional Neural Network) trained on ImageNet For the third experiment, we download and use a pre-trained model: Google's image classification architecture *Inception-v3* (Szegedy, Vanhoucke, et al., 2015) trained on ImageNet (J. Deng et al., 2009), a

¹⁰subsection 2.3.10: *Gradient descent and backpropagation*

¹¹subsection 2.3.15: *Convolutional Neural Networks (CNN)*

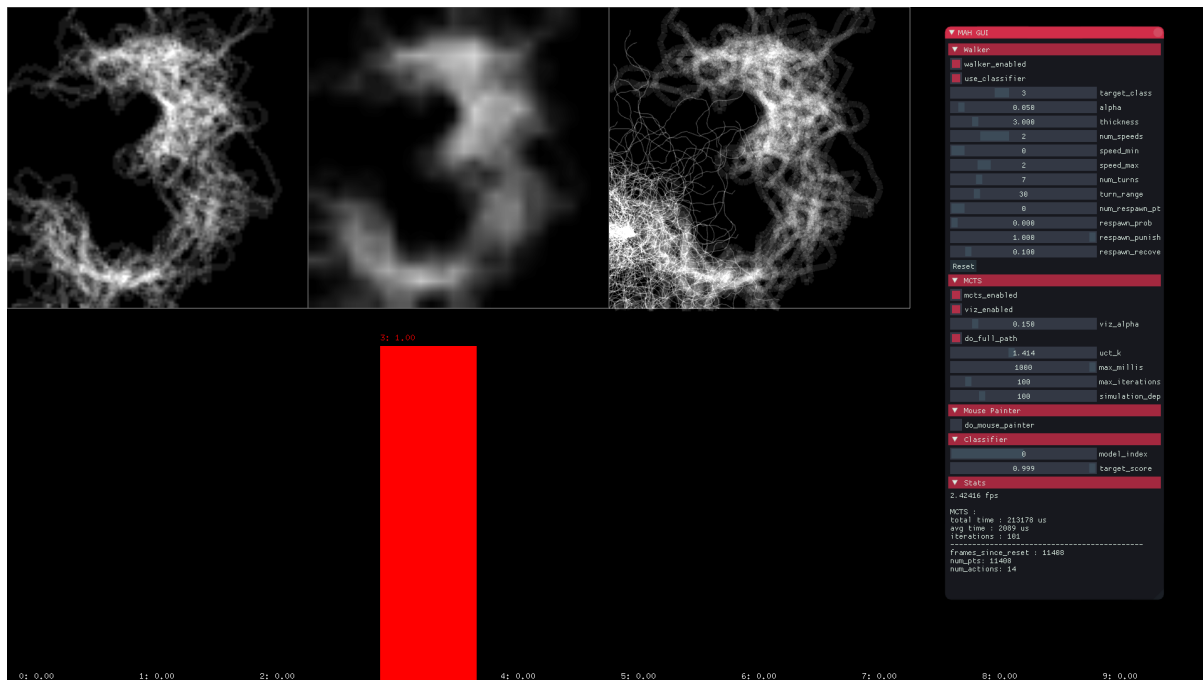


Figure 3.2: A screenshot from our software, running the MLR model trained on MNIST. The top left panel shows the current state of the canvas for the current timestep, the top middle panel shows the scaled down drawing used for evaluation, and the top right panel shows a visualization of all of the current MCTS rollouts for the current timestep. At the bottom of the screenshot, the class probability distribution shows that the classifier returns 100% confidence that the current state of the drawing is the digit ‘3’.

dataset consisting of millions of labelled images classified into one thousand classes. This model reached as low as 3.58% error in the 2012 ImageNet validation set for Top-5 error and was state-of-the-art at the time.

3.2.5 Results and discussion

As a result of these experiments, we observed that our system seemed to perform well, but did not always produce the results that we would expect or hope for. This unearthed some very interesting artefacts with regards to how CNNs learn to classify images. Our key finding with regards to these unexpected results, is that *discriminative* CNNs that are trained for image *classification*, are not well suited to *generative* applications in this manner, as they provide too many false positives. This in fact mirrors the findings of Nguyen et al. (2015), which was published while we were conducting our own research. The researchers use a very different method to ours. They use *evolutionary algorithms* to optimise for images that maximize class probabilities. For this reason, the aesthetic qualities of the images that their system produces is very different to ours. However, the conclusions are the same: there exists a space of images which are very ‘unnatural’, and would be classified as ‘random’ by humans, but are classified by CNNs *with very high confidence*, as belonging to specific classes.

Fig. 3.2 shows a screenshot of our software, running the MLR model trained on MNIST. In this particular case, we set the desired class to ‘3’. As can be seen in the panel in the top left, the current state of the agent’s drawing clearly resembles the digit ‘3’. The top right panel

shows all of the MCTS rollouts for the current timestep, i.e. these are the simulated, *imagined* paths. Every single one of the rollouts are individually evaluated by classifier, and the class probability is used as a reward for the action that led to that path. Along the bottom of the screenshot, red and blue bars visualize the distribution of class probabilities, i.e. the classifier’s *confidence* that the current state of the drawing belongs to each of the classes. In the video it can be observed how these rise and fall as the agent draws on the canvas and the drawing evolves. However, in Fig. 3.2, there is only one single red bar visible, because the classifier returns 100% confidence that the drawing resembles the digit ‘3’.

Of important note, is that the classifier here is not a *deep* Neural Network. It is a softmax regression on a linear transform. So as mentioned previously, as a *classifier*, this model does not perform optimally. However, as we use it in this *generative* context, it is able to guide the agent to create images resembling the desired class.

Using a more complex, deep architecture however, gives very different results. Fig. 3.3 shows a screenshot of our software, running the Inception-v3 model trained on ImageNet. In this particular case, we set the desired class to ‘meerkat’. Most people would probably think that the image in the top left panel is some kind of noise, and is quite far from resembling a meerkat. So it seems our system has failed in this case. However, interestingly, looking at the class probability distribution along the bottom of the image, it can be seen that the classifier is very confident — in fact with 100% probability! — that the image drawn by the agent is indeed a meerkat.

Similarly, looking at Fig. 3.4, we can see the results of running our system using the same classifier, where we set the target class to ‘white wolf’. Most people would again probably think that the image is a different kind of noise, and does not resemble a wolf at all. However, the classifier is still very confident — with 98% probability in this case — that the image is in fact a white wolf.

The failure in both of these cases, and all of the experiments that we ran using Inception-v3, is not a failure of the planning performed by MCTS, or of the way that MCTS is integrated with the classifier, or the backpropagation of the classifier confidence as a reward signal. In fact, the MLR study is a good demonstration that the overall system architecture and concept works. The failure in the latter cases, is because Google’s very deep model trained on ImageNet, is *incorrectly classifying* these random-looking textures as the desired class, with almost 100% confidence. In other words, the classifier is feeding an incorrect positive reward signal into MCTS. This is causing the agent to wander around the canvas in a manner that appears to humans to be random noise, but is classified by the network with close to 100% confidence to be the desired class.

After many tests like this, it became clear that as image classification models become more complex and accurate at *classifying* images, they became *worse* for use in a generative system as this one. *Discriminative* models with deep, complex architectures such as Inception-v3 are trained to classify images with high levels of translation, scale, rotation and noise invariance. Thus they generalize very well, and can classify *natural* images correctly with very high accuracy and confidence. However, as an undesired side effect, they also learn features which encourage them to *incorrectly* classify *unnatural* images — such as very particular distributions of noise and

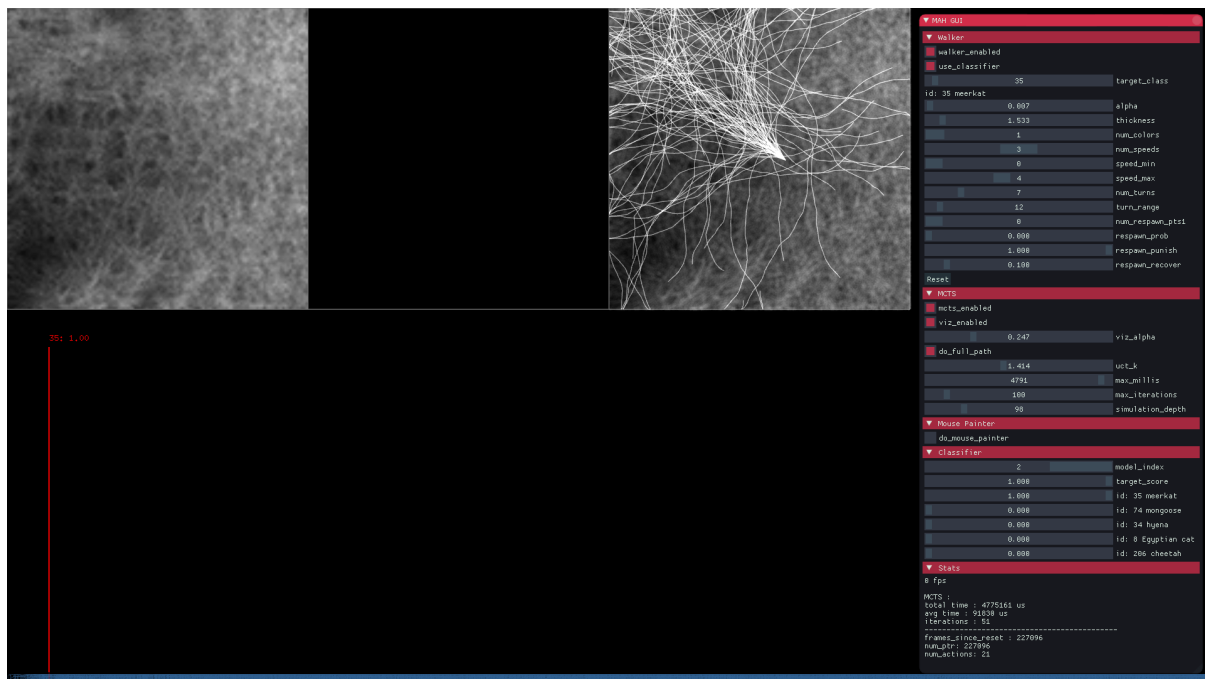


Figure 3.3: A screenshot of our software, running the Inception-v3 model trained on ImageNet with desired target class set to ‘meerkat’. At the bottom of the screenshot, the class probability distribution shows that the classifier is 100% confident that the drawing is of a ‘meerkat’.

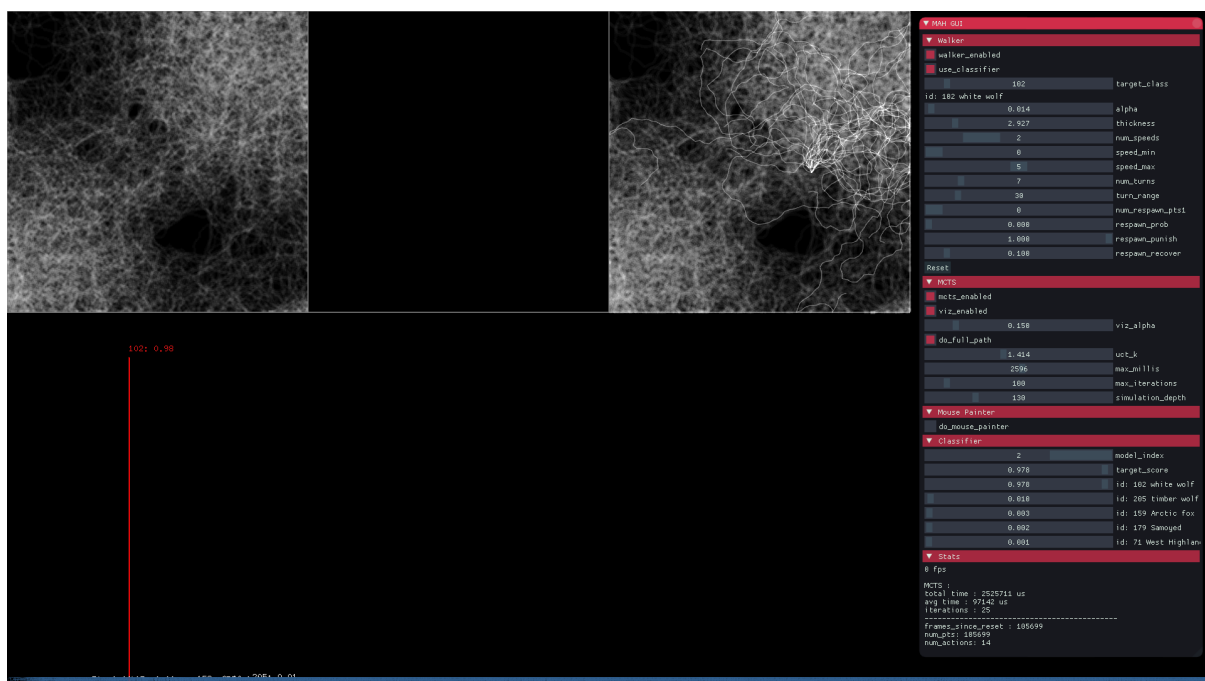


Figure 3.4: The same configuration as Fig. 3.3 except the desired target class is ‘white wolf’, and the class probability distribution shows that the classifier is 98% confident that the drawing is of a ‘white wolf’.

abstract shapes — with equally high confidence. In other words, the manifolds of the classes successfully capture the space of associated natural images, but they also include undesired, unnatural images as well. They produce a lot of false positives.

As a result of this, our generative system was not able to produce the kinds of images that we were hoping it could produce. For this reason, we did not invest more time into designing and developing modes of interaction which provide Meaningful Human Control, or Realtime Continuous Control. Instead, we chose to rethink our approach, and focus on an alternative study, which we discuss in the next section.

3.3 Realtime interactive text generation with RNN ensembles

3.3.1 Introduction

In the previous section, we presented a method for collaborative generative sketching, whereby we combine MCTS with a Deep Neural Network classifier. Our goal, was to create a generative system which continually produces some outputs (in this case, drawing sketches), and at any time, a person can meaningfully interact with and guide the system with Realtime Continuous Control. While our system worked overall as intended, it was let down by the fact that a *discriminative* Deep Neural Network returns too many false positives for it to provide reliable feedback.

With this in mind, in this section we investigate the use of a *generative* sequence model, in a similar Realtime Continuous interactive setting. Instead of continuing to work with images, we choose to perform this study with generative *text*. This is due to a number of reasons. First, in order to build a generative sequence model of images, we would need a suitable labelled vector-based dataset, and this was not available at the time of our research. Google’s Quickdraw dataset would not be available for more than a year (*Quick, Draw! Dataset*, 2017). Second, as we discussed in section 2.2.7: *Char-RNN*, character-based generative text models were becoming very popular. However, the tools available at the time did not provide any form of Meaningful Human Control over the text generated, let alone in a Realtime Continuous manner.

For this reason, we adapt our question to the domain of interactive *character-based text* generation. Our aim is again to create a generative system that continually produces some outputs, in this case not drawing but producing text character-by-character, while allowing a user to meaningfully interact with the system and guide it with Realtime Continuous Control. We again take the idea of *visual instruments*, as we described in section 1.4: *Visual instruments: Realtime Continuous Control*, as very strong design guidelines. And we try to design the system such that *users* are more like *performers* that *play* the system.

Our paper on this study was accepted to the *Recurrent Neural Networks Symposium* at the *30th Annual Conference on Neural Information Processing Systems (NeurIPS 2016)* as a poster presentation (Akten & Grierson, 2016b), and our prototype was accepted to be a live demonstration on the general NeurIPS conference demo track.

3.3.2 Background

For this study, we use an ensemble of Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) RNNs. More information on these architectures can be found in subsection 2.3.19: *Recurrent Neural Networks (RNN)*. In particular, our implementation is based on the character-based text model as described in Graves (2013), which is also the foundation for Karpathy’s popular Char-RNN.

RNNs are a type of neural architecture with cyclic connections. These allow the network to learn temporal patterns, which in turn allows RNNs to be very effective for sequence modelling. Notably, Graves demonstrates the use of LSTM RNNs to generate many different types of sequences, one of which is generating text character-by-character (Graves, 2013).

An LSTM RNN has an *internal state* which can be represented as a tuple of $(\mathbf{h}_t, \mathbf{c}_t)$ where \mathbf{h}_t and \mathbf{c}_t are the *hidden state* and *memory cell* vectors respectively at timestep t . At every timestep, these values are fed back into the network, and this effectively allows the network to ‘remember’ events from the past, and use that information to affect the predictions it makes for the next timestep. In other words, the predictions made by an RNN, is conditioned on the input, and the RNN’s internal state at that point in time.

In the case of character-by-character text generation, this equates to an RNN which outputs a probability distribution over all characters, given previous characters. To generate text, we sample this distribution to generate a character, and then feed this character back into the RNN to update its internal state. Repeating this procedure, we can generate long pieces of text.

However, the typical workflow at the time of our research, follows the same patterns that we describe in section 1.3: *Meaningful Human Control* under the subsection *Pressing a button*. This can be summarised as: i) collect some amount of data, ii) train a model on the data, and iii) press a button or run a script to produce a large chunk of text in the style of the training data.

With this workflow, there are two ways in which a person can influence the output of a Char-RNN model. The first is through the curation of training data. A Char-RNN model trained on the works of Shakespeare will generate text similar to Shakespeare, while a model trained on C code will generate text similar to C code. But once a model is trained, a Shakespeare model will not be able to generate text similar to C code or vice versa.

Regarding generative RNNs, there is another method of influencing the output, which is more subtle. This is known as *priming*. When a forward pass is run on a trained, but unprimed Char-RNN, its internal state will be uninitialized to begin with, so the distribution output will be unknown, and most likely random. Sampling a character from this distribution will be also be random, from an unknown distribution. To *prime* a model, we simply run forward passes on the trained model with each character of a *seed* text. Each forward pass with the next character from the seed, updates the internal state of the model. Once the entire seed has been fed into the model in this way, the model has been *primed* to continue from the seed. In other words, when the next forward pass is run, the distribution output will be biased towards continuing from the seed text. For example, if the Char-RNN is first fed the individual characters from ‘Good Mo’, this will set the internal state such that, when a forward pass is next run, it will output a distribution where the letter ‘r’ will have a high probability.

This is an effective way to influence the output of generative RNNs, and grant some level of control to a user. Using these methods, artists such as Ross Goodwin¹² and Allison Parish¹³ have created a number of influential text-based artworks.

However, the tools available still do not allow the level of control that we seek, especially with regards to *Realtime Continuous Control*, and thinking about *visual instruments*.

The question that we set out to investigate with this particular study is: *Is it possible to control the output of text, and change its style in realtime, whilst the text is being generated.*

¹²<https://rossgoodwin.com/>

¹³<https://www.decontextualize.com/>

3.3.3 Overview

We will first describe the functionality that our solution provides, as we believe this to be more important of a contribution, compared to how we actually implemented it. We will briefly summarise our implementation in the next section.

Our interactive Char-RNN ensemble software is designed to run across two displays presented side by side. This original layout can be seen in Fig. 3.5. To aid reading in print, Fig. 3.6 shows the two screens stacked vertically. On the left we have the *interaction screen*. This is displayed on a touchscreen monitor to aid seamless interaction. On the right we have the *output screen*. This is where the text is displayed as it is being generated, character-by-character.

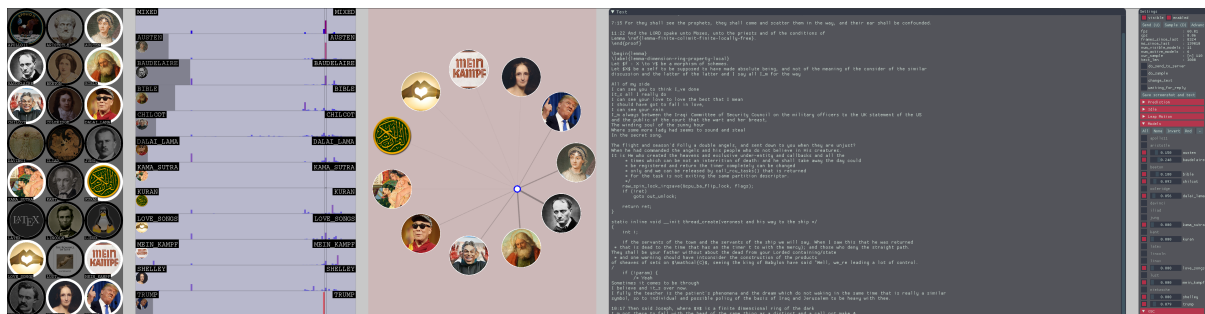


Figure 3.5: Screenshot from our dual-screen interactive Char-RNN ensemble software. On the left is the *interaction screen* which we typically display on a touchscreen monitor. On the right is the *output screen* where the text is displayed as it is generated character-by-character.

Styles

The user is presented with two dozen icons representing text *styles*. These can be seen in the top leftmost panel in Fig. 3.6 under **Style activation**, and they include a wide range of texts including the works of Shakespeare, Jane Austen, Mary Shelly, Friedrich Nietzsche, Charles Baudelaire, Carl Jung, texts from the Dalai Lama, the Iliad, love song lyrics, the King James Bible, the Chilcot Report of the Iraq Inquiry, cooking recipes, C source code for the Linux kernel, \LaTeX code, and many more. By clicking on these icons with a mouse or finger, the associated style is *activated* or *deactivated*. When a style is *active*, its icon appears in the **Style weights sliders** and **Style weights gestural interaction** panels.

Internally, all styles are always loaded into memory. This allows for the realtime activation or deactivation of styles with no time delay or overheads, or any negative impact on the momentary performance of the software. Potentially, loading too many styles could have an impact on memory requirements. However, with two dozen styles loaded, we saw no issue on a medium-level laptop at the time of this research in 2015. Temporarily activating or deactivating a style from this panel, simply adds it to (or removes it from) the interaction panels. This allows the user to reduce clutter on screen and simplify the interaction by temporarily deactivating undesired styles.

When at least one style is active, the system starts automatically generating text. This can be seen in the **output** panel in Fig. 3.6. This text appears character-by-character, at approximately 10-20 characters per second. It is reminiscent — and symbolic — of a person typing on a typewriter.

3.3. REALTIME INTERACTIVE TEXT GENERATION WITH RNN ENSEMBLES

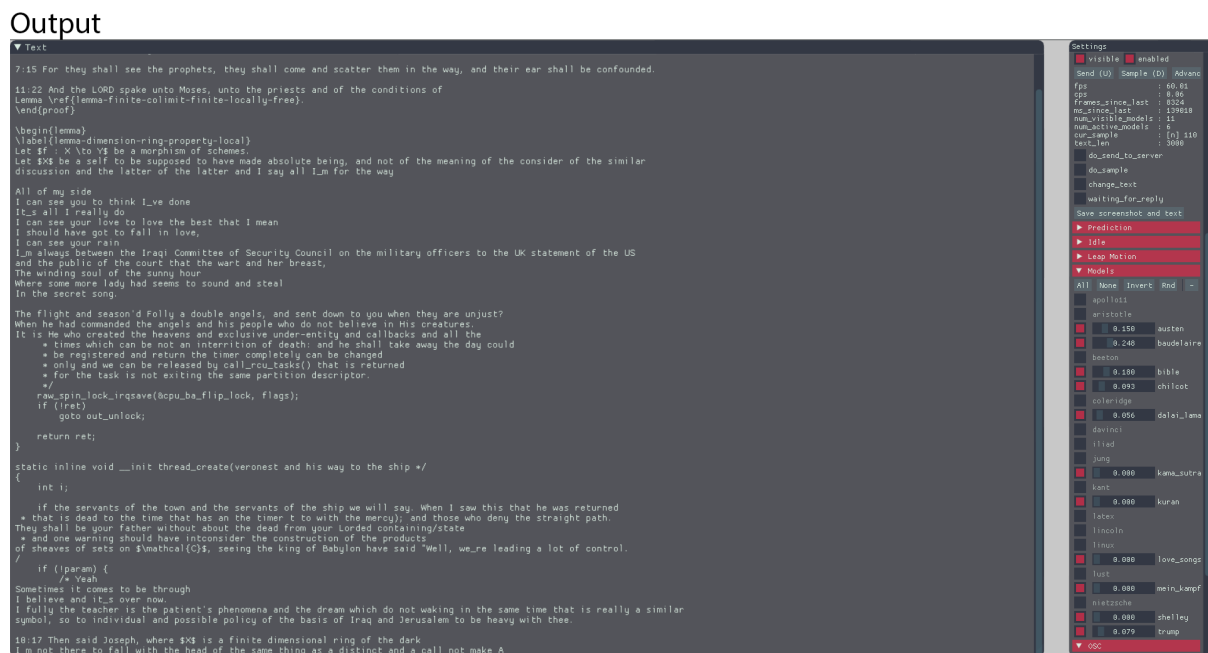
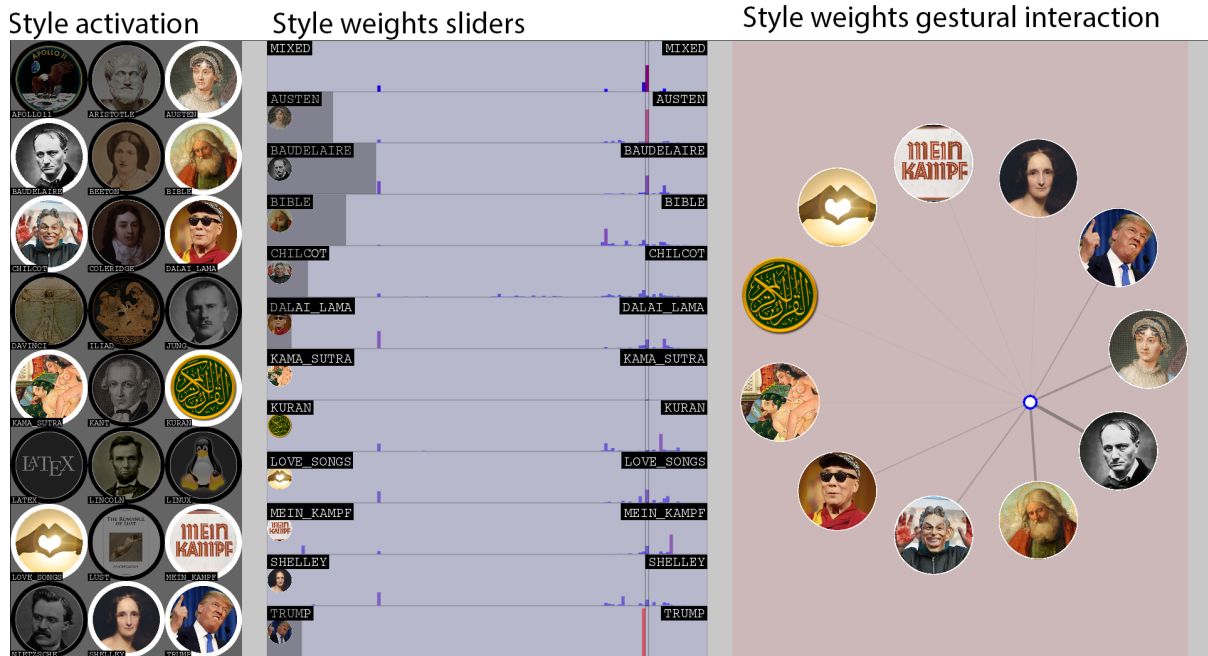


Figure 3.6: The two screens from Fig. 3.5 stacked vertically for ease of reading in print.

Style weights sliders

While the system is generating text in this manner, the user is able to adjust *weights* for each style via the onscreen **Style weights sliders**. These can be seen in top centre panel of Fig. 3.6. For example, while the system is generating text in the style of the *Bible*, the user can start mixing in a little bit of *Mary Shelley*, and then mix in a little bit of the poetry of *Coleridge* etc. This does not affect the text that has been generated so far. It only affects the new text that is being generated at that moment. In this way, styles can be changed mid-sentence, even mid-word.

Styles can be swapped fully. Bringing a new style in at 100%, will bring all of the other styles down to 0%. Again, this can be performed at any time. So a sentence that starts in the style of *Shakespeare*, can end in the style of *Baudelaire*.

Styles are not mutually exclusive, they can also be *mixed*. For example, a memorable outcome occurred when both the *love song lyrics* and *Linux C source code* styles were mixed together equally at 50%. Amongst the text that the system generated, was the phrase “static unsigned love”, a mix between “static unsigned long” — a common C expression — and of course, the word “love”.

The sliders allow very precise control over the weights of the styles. For example, it would be possible to set one particular style to 23%, and a second style to 77%. However, we have observed that this level of precision does not have a noticeable effect on the outcome. In our own testing, we observed that relatively large, roughly 25% increments in weights produced noticeable changes in the text generated. When the weights were modified in smaller increments, we couldn’t notice any difference in the output. We qualitatively think of using such large increments as *off*, *a little bit on*, *halfway on*, *mostly on*, and *fully on*. For this reason, since incredibly fine precision is generally not necessary, we also implement **gestural interaction**, which compromises precision for expressivity.

Style weights gestural interaction

The **gestural interaction** panel allows more *performative* control over the weights of the styles. Here, the active style icons are arranged in a circle as can be seen in the top right panel in Fig. 3.6. Moving around in this space using the mouse cursor or a finger touching the screen, the user can more expressively perform the manipulation of style weights. The weights are calculated based on the distance from the cursor, to each of the icons. This of course does not allow as fine control as using the sliders directly, but we believe that this is a negligible price to pay given the expressive freedom it adds. Especially when we consider that, as we mentioned in the previous section, high precision is not really necessary, and only large changes in weights are noticeable.

Furthermore, using the sliders requires the user to concentrate more on the interaction screen, and in particular on the weights sliders area. This is to ensure that their finger or mouse cursor is on the correct slider. However, the *gestural interaction interface* allows the user more freedom to look elsewhere, for example they can stay focused on the *output screen*. This is because with the gestural interface, they only need to be aware of *roughly* where their finger is on screen. This enables the user to more freely and expressively *perform* the broad strokes of

which styles to bring in, *when*, and at what *approximate* levels; without having to pay such close attention to the interaction screen. This makes the gestural interaction model more suitable for *performance and expressive interaction*, as we describe in section 1.4: *Visual instruments: Realtime Continuous Control*. In this case, we can think of the *user*, as more of a *performer*.

LeapMotion interaction

Going one step beyond the gestural interaction panel, we also developed a *LeapMotion* interface. LeapMotion (LeapMotion, n.d.) is a small, infrared sensor which is able to track hand and finger pose information in 3D space. Our interactive Char-RNN software also includes LeapMotion control, whereby instead of using a mouse cursor, or a finger on a touchscreen display, the user can simply move their hand in 3D space. We track the tip of their index finger, and its position in space is mapped to the screen to control the weights in a similar fashion to the gestural interaction panel. Just as it was in the gestural interaction panel, very precise position of the hand in space is not very important. Rather, it is the *approximate* position in space, and *when* the hand was moved to that position, which is most relevant. Reflecting on the analogy of performing with musical instruments as we describe in section 1.4: *Visual instruments: Realtime Continuous Control*, this mode of interaction further reinforces the notion that this is not a *user using* our system, but more like a *performer playing with our system* to perform the generation of text. In this particular case, it could even be described as a conductor *conducting* the generation of text.

We developed this interaction model in response to conversations with the performer and experimental poet Jennifer Walshe¹⁴. We imagined a live performance whereby text being generated by the system is read out loud via a text-to-speech system. In this case, we can remove the requirement for a screen altogether. This frees the performer to move and act much more freely. With a mode of interaction such as the LeapMotion interface, the performer does not have very precise control over the weights. However, given that the noticeable effectiveness of the weights can be thought of approximately and qualitatively as *none*, *little*, *half*, *a lot*, and *full*; this level of control can be learnt relatively quickly through building an intuition of how specific points in space correspond to specific styles. For example, “hand near right shoulder” might be one style on *full*; while “hand near left shoulder” might be another style on *full*; and in-between the two shoulders, styles are mixed accordingly.

Hearing the output of the system spoken by a text-to-speech interface, the performer can expressively respond with their body to control, steer and *conduct* the output of the system in realtime, without the burden of having to sit in front of a computer and look at a screen.

We discuss the significance of the LeapMotion interface, along with more ideas in section 3.4: *Conclusion*.

¹⁴<https://milker.org/>

3.3.4 System description

We implement our software by training a number of different character based LSTM models on different datasets. Given a *seed* text, each model outputs a probability distribution over all characters, from which the next character is sampled. We run each model on a common input, and we mix the probability distributions that they output. This can be thought of as an *ensemble* of generative RNNs, collectively outputting a *conditional probability matrix* at every timestep. Using the style mixture weights as *style probabilities*, we calculate a *joint probability distribution*, from which we sample the next character. We provide more details in our paper (Akten & Grierson, 2016b).

Our software consists of two main components: i) a python-based console backend **Server** which is responsible for loading and running all of the models, and ii) a C++ frontend interactive **Visualiser**. The two run as separate processes, and communicate via UDP using the OSC protocol. A summary of this architecture can be seen in Fig. 3.7.

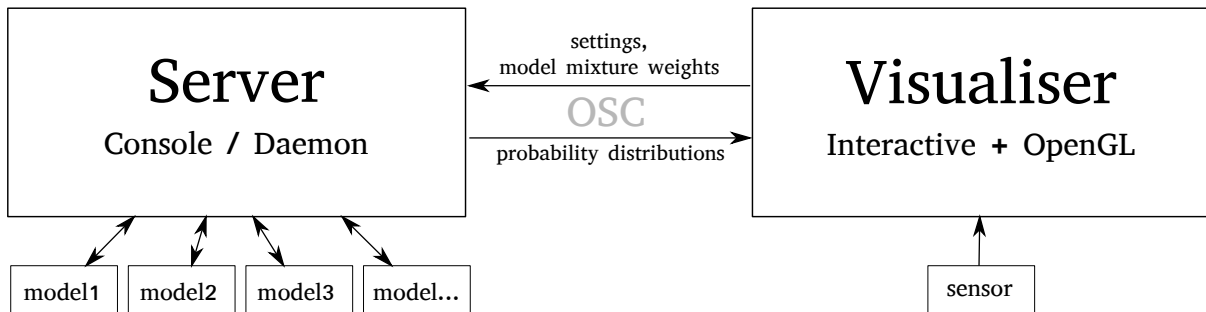


Figure 3.7: Software architecture for our interactive Char-RNN. The *Server* is a python-based console application with no Graphical User Interface, that manages all of the models. At every timestep, the Server receives a *seed* text from the *Visualiser*, runs it through all of the models, and returns all of the probability distributions output from each model to the Visualiser. The Visualiser is an OpenGL application which provides the visual interface and interaction. The screenshots in Fig. 3.5 and Fig. 3.6 are from the Visualiser. The Visualiser continually tracks the performer’s actions, either via mouse input, tracking the performer’s hands using a LeapMotion device, or via faders on an external midi controller. At every timestep, the Visualiser uses these inputs to calculate and update style mixture weights, which are used to calculate the *joint probability distribution* for the next character. The Visualiser samples the next character from this distribution, updates the screen with this new character, and also sends it back to the Server via OSC, so that the Server can update the internal state of each of the models.

We separate the two processes in this manner as it provides a number of benefits.

It is essential to separate the DNN related tasks, and the graphics and interaction related tasks into separate *threads*. For an interactive performative system such as ours, it is very important to maintain a stable, high framerate, so that both the animations, and the interaction always feels stable and fluid. For this reason, we ensure that our Visualiser always runs at a fixed 60fps. Feeding data into models and running inference can be a slow process, taking a variable amount of time depending on the number of models that are active. We do not want our visual and interactive experience to suffer, slow down, or feel sluggish when the models are being run. For this reason, it is essential to separate them into separate *threads*. Separating them into separate *processes*, effectively gives us separate threads by default, while providing other advantages.

Using separate *processes*, we can leverage the strength of different software development

environments and frameworks. For example, we use python and *Tensorflow* (Abadi & Others, 2015) for the backend Server managing the DNN related tasks, and we use the C++ creative programming toolkit *openFrameworks* (Lieberman et al., 2016) for the interactive frontend, as it provides a lot of functionality related to graphics, interfaces and interaction. This greatly speeds up and aids development.

Finally, separating the processes in this way, allows for more flexibility in the deployment of such a system. We have generally run both processes on the same computer, one with a GPU powerful enough to run all required models simultaneously. However, we can imagine a situation whereby we have a lightweight *client* computer running the Visualiser, and a separate, much more powerful GPU Server running the backend. This GPU Server could even be in the cloud, while the client can be any computer with an internet connection. Migrating to a setup such as this, involves little more than providing the IP address of the Server, to the client computer running the Visualiser.

3.3.5 Results and discussion

When we were previously discussing the concept of *mixing styles*, we mentioned an interesting outcome that we had observed. When we mixed the *love song lyrics* and *Linux C source code* models equally, at one point our system produced the text “static unsigned love”, a mix between “static unsigned long” and the word “love”. In our paper (Akten & Grierson, 2016b), we discuss more of these kinds of examples, and other notable outcomes that we observed. Since these are specific to the case of *character* based sequence models, we will not detail them in this thesis. Instead, in the conclusion of this chapter, we will reflect on our findings of this study in the context of *Realtime Continuous Meaningful Human Control*.

3.4 Conclusion

In this chapter, we presented two studies, both of which were investigating the same high level goal: generative systems which *continually* and *incrementally* produce some kind of output, while allowing a person to *meaningfully interact* with and guide the system with *Realtime Continuous Control*.

Our first study investigated this in the context of *drawing or sketching*. We implemented an agent-based method, using MCTS to guide the agent, and a discriminative CNN to evaluate the agent’s drawings and provide a reward signal. However, due to the amount of false positives that *discriminative* CNNs produce, this system was not able to produce the results that we were hoping for.

Instead of working with *discriminative* models, we switched to *generative* models, and we adapted our question to the domain of *text*. We developed a software based on an ensemble of LSTM RNNs, that allows Realtime Continuous interaction with this system, to influence the *style* of the text being generated. This interaction is possible via a number of modes, ranging from very precise manipulation of *sliders*, to more *expressive and performative gestural interaction* tracking the hand of the user in space with a LeapMotion sensor. In this latter case, we think of the user more as a *performer playing the system like an instrument*, or even a

conductor conducting the generation of text.

We describe why we believe in the importance of this kind of an approach in section 1.4: *Visual instruments: Realtime Continuous Control*. While this is an active area of research within the broader field of *ML* and performance, particularly within music as we mention in subsection 2.2.9: *Machine Learning for Artistic, Expressive Human Computer Interaction (AE-HCI)*, we are not aware of such studies at the time using state-of-the-art DL algorithms.

We do not claim that we have developed the most effective solution or interface for such a problem. Neither do we claim that our system provides optimal usability. Our aim with this research is two-fold.

First, we are investigating how we can introduce points of interaction into DL systems that currently do not allow any form of meaningful or Realtime Continuous Control.

Second, and more crucially, we are hoping to encourage more conversations and thinking in this direction. We were very happy to be accepted to show a live demonstration of our interactive Char-RNN software at NeurIPS 2016, to thousands of researchers. Having spoken to countless researchers at this conference trying our software, we can summarise the responses as ranging from “This is so awesome”, to “I don’t understand what this is for. What is the application for this?”. Clearly not everybody shares our priorities. However, many people do. And we would like to help shift thinking in that direction.

Google’s highly successful *Sketch-RNN* and related web applications perfectly address the questions and motivations which we raise in our first study (though with a very different, and more successful technical approach). The popularity and success of very recent research, also looking into Meaningful Human Control over generative Deep Neural Networks (Park et al., 2019; Karras et al., 2019; Simon, 2019; Bau et al., 2019; Karras et al., 2020; Härkönen et al., 2020; Jiang et al., 2020; Broad et al., 2020) is also a prime example of this.

For the interactive Char-RNN study, we developed the LeapMotion interface, not because we believe this is the best mode of interaction for this medium. We developed it because we believe this is a starting point, that hopefully other researchers can follow through on. It acts as an example to demonstrate the vast number of possibilities with regards to interfaces for Deep Neural Network manipulation, an area which we believe is highly under-researchers.

For example, we also considered detecting facial features and mapping them to style weights. In other words, we imagined that different kinds of facial expressions could control the style of the text generated. A fully smiling face could correspond to the *Dalai Lama* style, while a frowning face could correspond to *Trump*. In a similar fashion, different styles could be assigned to eyebrows raised, lowered, looking angry, eyes closed, open etc. And then through various facial manipulations, the style of the text is altered in realtime. In conclusion, the interactive Char-RNN software that we developed, demonstrates perfectly what we were hoping to demonstrate with this study.

After this, we move onto a more complex and high-dimensional domain, *pixel-based images*. This will be the medium that we work with in the remainder of this thesis. We start in the next chapter looking at how image-based Deep Neural Networks learn. For this, we develop an interactive training software that trains in realtime on a live camera feed.

Chapter 4

Hello World: Realtime interactive training as an informative and performative tool

4.1 Introduction

In this thesis, the two key points that we prioritise when thinking about the design of our interactive generative systems, are *Meaningful Human Control*, as we describe in section 1.3: *Meaningful Human Control*; and *Realtime Continuous Control*, as we describe in section 1.4: *Visual instruments: Realtime Continuous Control*. In particular, we think of these systems that we are trying to build as *visual instruments*, where the *user* can be thought of as more of a *performer* that *plays* the system.

In the previous chapter, we examined two different approaches to this topic. The first of these studies looked at *sketching*, while the second looked at character-by-character generation of *text*. The second study that we presented, section 3.3: *Realtime interactive text generation with RNN ensembles*, achieved perfectly what we were hoping for, demonstrating the potential of such an approach. Our system provides both Meaningful Human Control, and Realtime Continuous Control. And we were able to gesturally *conduct* the generation of text, using our body in an expressive and performative manner, analogous to a visual instrument.

We began this inquiry in the previous chapter, working with data of relatively low dimensionality. Our *MCTS agent* selects actions from within a dozen or so options, and our *interactive Char-RNN ensemble* selects the next character from less than a hundred or so characters.

In this, and subsequent, chapters, we move onto and apply the same kind of thinking to data that has many orders of magnitude higher dimensionality. We work with pixel-based raster images.

In this chapter in particular, we investigate how image-based deep generative models learn. We develop a software system that performs *realtime training* on a live video feed, while allowing for the Realtime Continuous manipulation of a number of hyperparameters such as learning rate, momentum, gradient clipping thresholds, loss function and many more ¹. Observing the results

¹A full list can be seen in subsection 4.4.1: *Hyperparameters*

of these hyperparameter manipulations, we are able to build a qualitative understanding of how they impact the training process.

Furthermore, an unplanned outcome of our system, is that it also provides great potential as a performative tool.

Manipulating hyperparameters in realtime while the model is training, produces aesthetically unique and interesting outputs, and effectively transforms the Deep Neural Network into a medium which we can play like a *visual instrument*.

In addition, the affordances granted by our software is not limited to the fact that we can manipulate hyperparameters in realtime while the model is training live. In fact the model is training on a *live video feed*. This introduces a second mode of interaction, a very expressive and performative mode of interaction. Namely, by manipulating our hands, body or objects in front of the camera while the model is training, we can observe the results this has on the system, and we can act accordingly. Any movements we make in front of the camera, effectively introduces new training examples into the training dataset. We can then make experimental gestures and movements in front of the camera, and observe how these new training examples influence the predictions that the model makes and the images generated as a result of this. Through this Realtime Continuous interaction with immediate feedback, we eventually build an understanding of how to meaningfully control this process, and we can leverage this new found knowledge and experience to use the system as a performative environment.

In this respect, combining both of these modes of interaction, we use our system not only for technical research, for building a qualitative understanding of the effects of hyperparameters or subtle changes to training examples. But we also used it for artistic, aesthetic and performative research. We will build upon this aspect of the software, focusing on these two modes of interaction that we just mentioned, more in the following chapter chapter 5: *Learning to see: Digital puppetry through realtime video transformation*.

Finally, due to the fact that our system demonstrates and sheds light on how Deep Neural Networks learn, and is especially accessible by a non-expert audience, the system itself has garnered interest from arts and cultural institutions. As a result, we have exhibited the system itself as an interactive video installation at events and venues including Ars Electronica (Linz, AT, 2017), International Documentary Film Festival Amsterdam / IDFA (Amsterdam, NL, 2017), and the Moscow Museum of Modern Art (Moscow, RU, 2018) (Fig. 4.1). The work has also been included in MIT's Open Documentary Lab *docubase*².

4.2 Motivations

There are a few key moments that led to the development of this particular study that we present in this chapter.

In subsection 2.3.13: *Hyperparameter search*, we describe the importance of *hyperparameters*. These are variables that are not learnable via a typical training process, and they define or control the architecture of a DNN or various aspects of the optimisation process. These include variables such as the number of layers in a DNN, the activations functions and number of

²<https://docubase.mit.edu/project/learning-to-see>



Figure 4.1: **Hello World** at *The Moscow Museum of Modern Art*, 2018 (Photo © Yuri Palmin)

neurons in each layer etc. They also include variables such as the loss function or optimisation algorithm used, learning rate, momentum, gradient clipping thresholds, regulariser weights, and many many more.

An optimal configuration of hyperparameters is essential to ensure that training a DNN converges to a desired solution. However, as we mentioned, hyperparameters are not *learnt* via training, but have to be set manually. Typically, there are no clearly defined rules as to what values these hyperparameters should be set to. Instead, hyperparameter values are selected purely based on empirical experience, which may or may not be transferable from one domain or problem to another. For this reason, it is common to perform a *hyperparameter search*, whereby up to *hundreds* of models are trained on the same dataset, slightly varying hyperparameters by a small amount in each case. From these hundreds of models, the model which performs the best, i.e. produces the most accurate predictions on a validation dataset, is selected as the best model ³.

The difficulty and fragility of manual hyperparameter selection is a well known open problem in DL research. For this reason, under the umbrella of *meta-learning*, or *learning to learn*, researchers are investigating more robust, automated methods of hyperparameter selection (Andrychowicz et al., 2016; Finn et al., 2017; Zoph et al., 2018; Rusu et al., 2019).

In the meantime however, we are looking for ways to build a better qualitative understanding of how these hyperparameters affect training, and the solution that the model converges to. In particular, we are looking for ways of better understanding the effects of individual

³Alternatively, the top N models may be used as an *ensemble*. However, this is outside the scope of this thesis so we will not dig deeper into this.

hyperparameters.

When appropriate, we do also use the “train hundreds of different models with randomised hyperparameters” approach. And while this may be the current best method for selecting the most *quantitatively optimal* model, we do not believe it is necessarily the best method for allowing a human to *build a qualitative understanding* of how the hyperparameters influence the outcomes. Based on our experience with *visual instruments*, including both the study that we presented in the previous chapter, and including our experience outside of the work that we present in this thesis, we believe that a system that provides Realtime Continuous Control, and allows for experimentation of parameters with immediate feedback, is an essential complement to build such a qualitative understanding.

Our curiosity in this direction is further reinforced by the results of the sketching MCTS agent that we discussed in the previous chapter. In particular, the capacity for the CNN classifier to provide so many false positives, led us to inquire more about *how and what* DNNs learn. This is already a very active area of research (Erhan et al., 2009; Simonyan et al., 2013; Zeiler & Fergus, 2013; Mahendran & Vedaldi, 2014; Dosovitskiy & Brox, 2015; Nguyen et al., 2015; Olah et al., 2017; Mordvintsev et al., 2018; Olah et al., 2018). And as we described in the previous paragraph, we believe that a system which allows Realtime Continuous experimentation will provide very valuable complementary insights.

Finally, at the risk of stating the obvious, the data that a model is trained on, defines what the model will learn, and will ultimately produce. Training a generative DNN on images of cats, will result in a model that produces images of cats; while training a generative DNN on images of dogs, will result in a model that produces images of dogs. This should come as no surprise. However, an interesting question arises when we consider much more subtle variations to the training examples that we provide to a model. In order to investigate this, we again seek a Realtime Continuous system that allows for subtle realtime manipulation of training examples, and provides immediate feedback.

On top of all of these, which we can summarise as *technical* points of inquiry, we have our usual additional agenda, which is to investigate *deep visual instruments*: new modes of performative, artistic expression, using Deep Learning models as a medium and instrument.

For all of these reasons, we develop the realtime training software that we describe in this chapter.

4.3 Background

The Deep Learning architecture that we use in this study, is a *Variational Auto-Encoder* (VAE) with *Convolutional* layers. The Convolutional layers provide the architectural details that makes the Neural Network suitable for 2D pixel-based image processing. We discuss these in more detail in subsection 2.3.15: *Convolutional Neural Networks (CNN)*.

VAEs we discuss in subsection 2.3.17: *Variational Auto-Encoders (VAE)*. However, below we will summarise a few points that we will build upon in this chapter, and we will refer back to them in later sections.

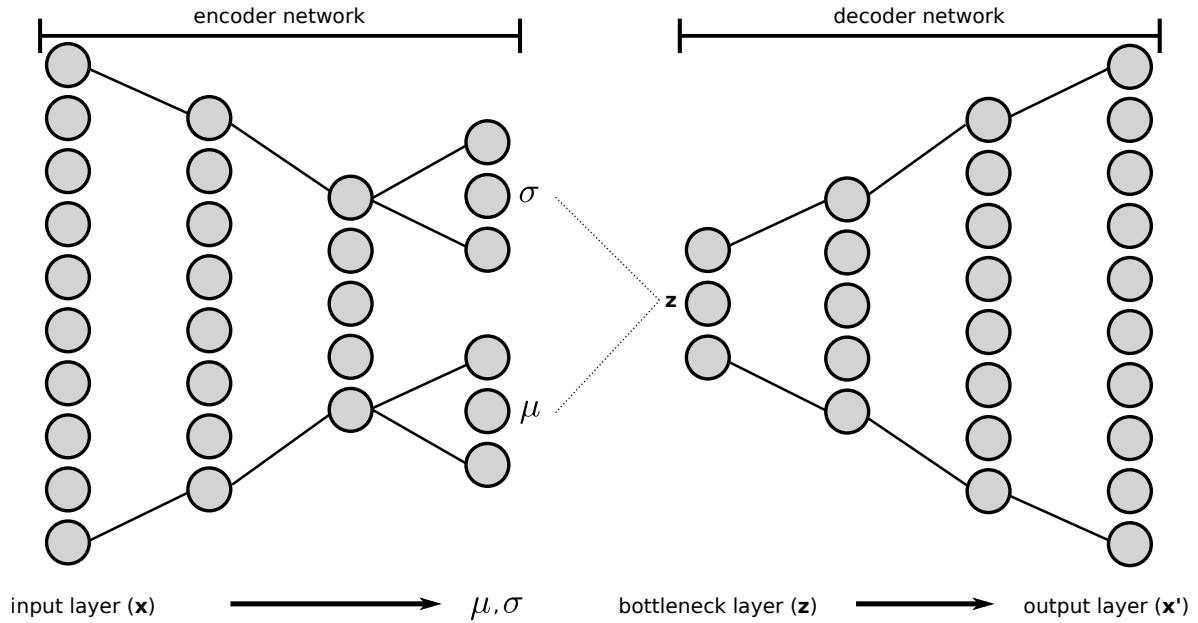


Figure 4.2: Variational Auto-Encoder. The output of the encoder network is the mean μ and standard deviation σ of a Normal distribution, from which a latent vector z is sampled $z \sim \mathcal{N}(\mu, \sigma)$. This is then fed into the decoder network for decoding.

1. An *Auto-Encoder* (AE) is a type of deep architecture whereby the model tries to effectively compress and then decompress its inputs — which are in this case, images from a live video feed. This is accomplished by a *bottleneck* layer which has much smaller dimensionality in comparison to the AE’s inputs. For example, a colour image of dimensions 256x256 pixels, might be compressed down to pass through a bottleneck layer of dimensionality 128. The aim of this compression is to encourage the network to identify and learn to extract salient features of the data, and learn more concise and optimal representations.
2. An AE can be thought of as consisting of two networks back-to-back. The first is an *encoder network*, which compresses an input x to the bottleneck layer. We think of this as the *latent representation* z . The second network is a *decoder network*, which decompresses the latent representation z from the bottleneck layer, to an output x' . Given suitable training examples, architecture and hyperparameters, the VAE should be able to produce x' such that $x' \approx x$.
3. A *Variational* Auto-Encoder is a type of AE that is also a *generative model*. It learns the distribution of data. This enables us to sample new images from this latent distribution, or perform meaningful latent manipulations. We discuss these in more detail in section 2.1: *Generative models*.
4. What enables a VAE to be a generative model, is the *parametrisation trick*. This is a *regulariser*, an additional term in the loss function, that ensures the distribution of the bottleneck layer (also known as the *latent distribution*) converges to a desired distribution. Typically, this is a standard normal distribution $\mathcal{N}(0, 1)$. In other words, we can sample a random $z \sim \mathcal{N}(0, 1)$ and feed this through the decoder to produce new images that resemble the training examples.

5. To be more precise, in a VAE, both the encoder and the decoder networks are probabilistic models. The encoder models the probability of $P(\mathbf{z}|\mathbf{x})$, while the decoder models $P(\mathbf{x}|\mathbf{z})$.
6. The loss function of a VAE consists of two components. The first component is the *reconstruction loss*, which ensures that the output of the VAE \mathbf{x}' resembles the input \mathbf{x} . We will discuss this in the next point. The second component is the *latent loss*. This is the regulariser used for the parametrisation trick, which ensures that the latent distribution converges to the standard normal distribution $\mathcal{N}(0, 1)$. This is typically the KL divergence between the latent distribution, and the standard normal distribution.
7. Typically, the reconstruction loss is a pixel-wise L1, L2 or Cross Entropy loss. It is well known that these produce blurry results (I. Goodfellow et al., 2014; Theis et al., 2016; Larsen et al., 2016). One approach to address this has been a completely alternative approach to VAEs altogether, in the form of *Generative Adversarial Networks* (GAN) (I. Goodfellow et al., 2014). We discuss this in subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*, and in the next two chapters we will investigate GANs. More recently, there have also been attempts at addressing this within VAEs (S. Zhao et al., 2017; Dumoulin et al., 2017; Rezende & Viola, 2018; Shu et al., 2018). In the work that we present in this chapter, which came before these more recent studies, we take an alternative approach, which is unique to the domain of images. We implement a loss function based on *Multi-Scale Structural Similarity Index Measure* (MS-SSIM) (Z. Wang et al., 2004) to achieve perceptually significantly superior results.
8. We have also observed that the relative weight of the reconstruction loss and latent loss is very important to how the model performs. For this reason, we introduce a new W_{KL} hyperparameter, which is a weight multiplier on the KL divergence loss term (i.e. the latent loss). At the time of our work, this was not mentioned in the literature, and was later independently investigated and confirmed (S. Zhao et al., 2017; Higgins et al., 2017; Van Den Oord et al., 2017).

It is worth noting again, as we have mentioned above, theoretical improvements to VAEs have been suggested (S. Zhao et al., 2017; Dumoulin et al., 2017; Rezende & Viola, 2018; Shu et al., 2018). And in fact, GANs have also been proposed as an alternative, although they lack efficient inference capabilities to encode existing data points to latent representations, and we discuss this in section 2.1.3: *Latent representation recovery*. However, the method that we propose in this chapter, is not strongly tied to the VAE architecture that we demonstrate it with. Instead, our software is more of a *shell* that can sit around potentially any neural architecture with suitable adaptations. In fact we have tested our software with both *unconditional* and *conditional* DCGANs, namely *pix2pix*. Documentation of such an adaptation can be seen in Fig. 4.3. However, at the time of our research, available consumer GPU hardware did not provide satisfactory realtime interactive performance when training GANs, and generally dropped below 1fps. For this reason, in this chapter we focus on VAEs, with which we are able to achieve 30–60fps.

4.3. BACKGROUND



Figure 4.3: An adaptation of *Hello World* at “Artists and Robots”, Astana Contemporary Art Centre, 2017

4.4 System description

We will begin by explaining what exactly our system does. Later in section 4.5: *Experiments and results*, we will discuss the affordances that this grants us.

In order to realise the above stated aims, we built a bespoke system that trains in realtime on a live camera feed, while allowing for the manipulation of a number of hyperparameters in realtime. Our system is built from the ground up in python using the Deep Learning framework *Tensorflow* (Abadi & Others, 2015) and *PyQtGraph*⁴. for frontend and visualisation. PyQt-Graph is an open-source Scientific Graphics and GUI Library based on Qt, designed for high performance, realtime applications.

Like many realtime systems, our system operates via an update loop. On a laptop with an Nvidia GTX 1070 our system runs at around 10-20 frames-per-second (fps); and 30fps–60fps on a desktop PC with an Nvidia GeForce GTX 1080 Ti. The exact figures depend on the hyperparameters and complexity of the Neural Network architecture that we set up.

When our application is launched, the model weights are initialised to random parameters using Xavier initialisation (Glorot & Bengio, 2010). Following the update loop detailed in Alg. 1, for every frame of the input video feed, our software performs one *forward* pass to run *inference* on the video input, it displays the generated images, and then runs one *backwards* pass to *train* and update the model weights. And this update loop is repeated indefinitely at realtime, interactive framerates.

This is a form of *online learning*, in which model parameters are updated dynamically, as new data streams in. Online learning is a well established area of research, and dates back to the earliest days of ML (Shalev-Shwartz, 2011). However, online learning is currently mostly limited to low dimensional data, using shallow models with convex objective functions. Applications of online learning with deep architectures is not common, although there is growing interest in the field (Sahoo et al., 2018).

It is important to underline how different this is in comparison to typical DL workflows. Within DL research, especially within the creative applications of DL, it is common practice to first train a model on some data, which can often take hours, days, weeks or even months depending on the amount of data and complexity of the architecture. And after this training phase, the model is used for inference.

With our method, every update loop of the application consists of one *inference* (or *forward*) pass, and one *training* (or *backward*) pass. Via a GUI, we can manipulate and experiment with a large number of hyperparameters while this *inference-training loop* is running, and we can observe the results in realtime. We do not argue that our method is capable of producing a more optimal model, compared to a traditional hyperparameter search consisting of training hundreds of models. However, we do claim, that our method is *complementary*, and can provide valuable insights into how these hyperparameters effect the training process and the solution that the model converges.

⁴<http://pyqtgraph.org/>

Algorithm 1: Hello World: Update loop

```

// Main variables, updated every frame
1 hyperparameters: hyperparameters which can be manipulated in realtime via the GUI;
2  $\Theta$ : current DNN weights;
3  $\mathbf{x}_{\text{live}}$ : current input image grabbed from the live video feed;
4  $\mathbf{z}_{\text{mode}}$ : latent representation for mode, where mode  $\in$  (live, perturbed, random);
5  $\mathbf{y}_{\text{mode}}$ : output image generated having decoded  $\mathbf{z}_{\text{mode}}$ ;

6 foreach update_loop do
7   hyperparameters  $\leftarrow$  read all hyperparameters from GUI;
8   architecture_changed  $\leftarrow$  CheckNNArchitecture(hyperparameters);
9   if architecture_changed then
10     BuildNNArchitecture(hyperparameters);
11      $\Theta \leftarrow$  initialise randomly with Xavier Initialization;
12   end

13    $\mathbf{x}_{\text{live}} \leftarrow$  grab current image from live video feed;

14   begin Forward pass with current weights  $\Theta$ 
15     begin mode Seeing: live video input
16     |  $\mathbf{z}_{\text{live}} \sim \text{Encode}(\Theta, \mathbf{x}_{\text{live}})$ ;
17     |  $\mathbf{y}_{\text{live}} \leftarrow \text{Decode}(\Theta, \mathbf{z}_{\text{live}})$ ;
18     end

19     begin mode Reminiscing: live video input perturbed with noise offset
20     |  $\mathbf{z}_{\text{perturbed}} \sim \mathcal{N}(\mathbf{z}_{\text{live}}, \text{hyperparameters}[\text{'noise\_amount'}])$ ;
21     |  $\mathbf{y}_{\text{perturbed}} \leftarrow \text{Decode}(\Theta, \mathbf{z}_{\text{perturbed}})$ ;
22     end

23     begin mode Dreaming: random sample
24     |  $\mathbf{z}_{\text{random}} \sim \mathcal{N}(0, 1)$ ;
25     |  $\mathbf{y}_{\text{random}} \leftarrow \text{Decode}(\Theta, \mathbf{z}_{\text{random}})$ ;
26     end
27   end

28   begin Backward pass to update weights
29   | loss  $\leftarrow \text{CalculateLoss}(\mathbf{x}_{\text{live}}, \mathbf{y}_{\text{live}}, \text{hyperparameters})$ ;
30   | gradients  $\leftarrow \text{CalculateGradients}(\text{hyperparameters})$ ;
31   |  $\Theta \leftarrow \text{Backpropagate}(\text{gradients}, \Theta, \text{hyperparameters})$ ;
32   end

33   begin Display in GUI
34   | Stimulus  $\leftarrow \mathbf{x}_{\text{live}}$ ;
35   | Seeing  $\leftarrow \mathbf{y}_{\text{live}}$ ;
36   | Reminiscing  $\leftarrow \mathbf{y}_{\text{perturbed}}$ ;
37   | Dreaming  $\leftarrow \mathbf{y}_{\text{random}}$ ;
38   end
39 end

```

4.4. SYSTEM DESCRIPTION

Fig. 4.4 shows a screenshot from our software. This is a full capture of the entire screen, although it excludes the GUI, as that is a separate window which we usually place on a second monitor. In the following pages, in order to save space on the page, we will crop the screenshots to show only areas relevant to the topic at hand.

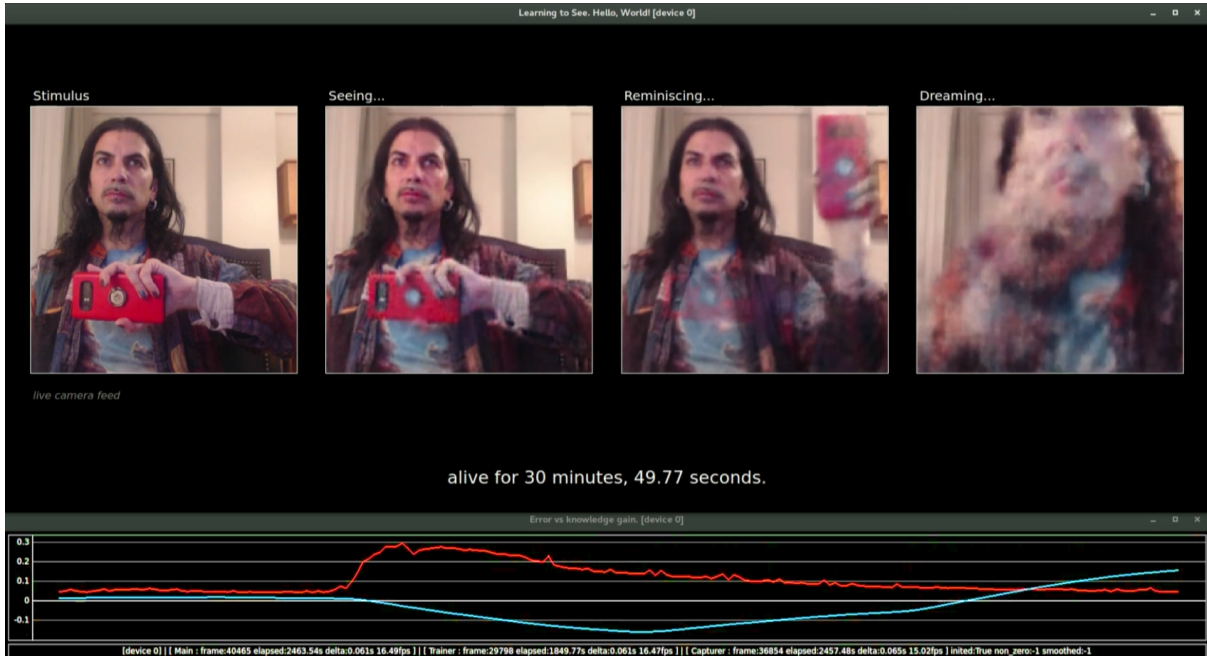


Figure 4.4: A screenshot from our *Hello World* software.

In this screenshot, there are four panels showing images labelled *Stimulus*, *Seeing*, *Reminiscing* and *Dreaming*. We use these terms as metaphorical representations of the operations that are being performed in each panel. The details of these operations can be seen in Alg. 1, and we summarise them below.

- The *Stimulus* panel shows \mathbf{x}_{live} , the live video feed that is presented to the model.
- The *Seeing* panel shows \mathbf{y}_{live} , a straight reconstruction of the live video feed. In other words, the current image from the camera \mathbf{x}_{live} , is fed through the encoder and compressed into a latent representation \mathbf{z}_{live} . This latent representation is fed through the decoder and decompressed to produce the image \mathbf{y}_{live} . This is what the model is ‘seeing’ in its ‘mind’, as a result of being presented with this stimulus.
- The live video feed’s latent representation \mathbf{z}_{live} , is additionally perturbed by a small amount of noise to produce a new latent representation $\mathbf{z}_{\text{perturbed}}$. Feeding $\mathbf{z}_{\text{perturbed}}$ through the decoder, a new image $\mathbf{y}_{\text{perturbed}}$ is generated and shown in the *Reminiscing* panel. Since $\mathbf{z}_{\text{perturbed}}$ is essentially a random position *in the local neighbourhood of* \mathbf{z}_{live} , and assuming that the space of latent representations is indeed well structured, we should expect the image generated as a result $\mathbf{y}_{\text{perturbed}}$, to be somehow similar — structurally and/or semantically — to \mathbf{y}_{live} . We discuss the reasons behind this expectation in subsection 2.1.3: *Latent manipulations*. This is an image that the model is being ‘reminiscent’ of, as a result of being presented with this stimulus.

- The *Dreaming* panels shows a ‘random’ image $\mathbf{y}_{\text{random}}$, generated by sampling a random $\mathbf{z}_{\text{random}}$ from the latent distribution $\mathcal{N}(0, 1)$, and decoding through the model. This is the model ‘dreaming’ random images that are not directly connected to the current stimulus, but is instead a result of the model’s history and life experience.

The **red graph** at the bottom of the screenshot is a plot of the *error*. This is the *loss* calculated between \mathbf{x}_{live} and \mathbf{y}_{live} , as per the loss function currently selected via the GUI.

The **cyan/blue** graph we call *knowledge gain*. This is loosely based on Jürgen Schmidhuber’s formal theory of *curiosity, creativity, compression* and beauty (Schmidhuber, 2006, 2007, 2009a, 2009b, 2010a, 2010b, 2012a). In this model, an agent’s subjective view on the *interestingness* of a phenomena is measured as the *first derivative of compression*. In other words, an *increase* in the agent’s ability to compress *previously uncompressable* data, is indicative of the agent learning about new regularities, and thus improving its model and knowledge of the world. In *Hello World*, we treat the *decrease* of error, as an *improvement* of the compressor, and thus an *increase* in *knowledge gained* about the world. We implement this by smoothing the error plot with a rolling window, and accumulating it’s negative derivative. This is by no means an accurate metric, as it does not take into account *changes in stimulus*. For example, swapping a novel stimulus for a previously seen stimulus will also reduce the error, without an increase in knowledge. There are ways in which this could be taken into account. However, this is far outside the scope of our thesis, so we implement this simple metric as an approximate indicator of *knowledge gain* for situations where we know the stimulus to be relatively static.

4.4.1 Hyperparameters

One of our primary motivations in building this system, is to allow for the realtime manipulation of hyperparameters, while the system is training. The hyperparameters which our software makes available for manipulation can be grouped into two categories.

Architecture hyperparameters define the architecture of the Neural Network. These hyperparameters can be modified while the system is running. However, this will reconstruct the computation graph, and the number of weights in the network is likely to change. Hence, changing any of these hyperparameters will result in the weights being reinitialised, and this will effectively reset the training. For this reason, we think of these as *destructive* edits. This is implemented in line 9 of Alg. 1. These hyperparameters include:

- Dimensions of the bottleneck layer \mathbf{z}
- Number of layers
- For each layer, convolution hyperparameters such as kernel window size, filter depth, and stride. We detail these in subsection 2.3.15: *Convolutional Neural Networks (CNN)*

Non-destructive hyperparameters do not affect the structure of the architecture, and can be modified in realtime without requiring a reinitialisation of the weights. In other words, the Neural Network can continue to train while we modify these values, and we can observe the results in realtime. Where these are *boolean* or *multiple-choice* options, our software constructs

multiple paths in the computation graph for each of the options, and seamlessly switches between them at runtime depending on the options selected by the user. In other words, there is a path in the computation graph for each loss function, optimisation algorithm, and any other option that is provided to the user. We implement the *real valued* hyperparameters as *tf.Variable* objects, and we feed their values from the GUI at runtime. These non-destructive hyperparameters include:

- Activation functions for each of the layers \in (tanh, sigmoid, ReLU, or Leaky ReLU)
- L1 and/or L2 regularisation amount on the weights
- Reconstruction loss \in (L1, L2, Cross Entropy, or MS-SSIM)
- A weight multiplier W_{KL} for the latent loss KL Divergence
- An option to toggle between a *variational* and a *normal* auto-encoder
- Optimiser \in (SGD with momentum, RMSProp, Adagrad, Adadelta, or Adam)
- Values for optimisation hyperparameters associated with the selected optimiser. These include learning rate, momentum, β_1 , β_2 etc.
- Values for gradient clipping thresholds, both on *L2 normals* and *per component*
- Batch size and options for exponentially decaying memory and data augmentation
- Noise amounts, injected at input and/or in latent space
- Video input device \in (USB cameras, network IP cameras, video files, or folders of images)

4.4.2 Batch size, exponentially decaying memory and augmentation

As we have previously mentioned and detailed in Alg. 1, with every update, the model is trained on a single image, which is the current image from the video feed \mathbf{x}_{live} . In this case, we observe that the model converges to reconstruct the current image very quickly, provided that the video feed does not change dramatically. However, the model ‘forgets’ previous images it has seen, and is unable to reconstruct images that were presented to it just a few seconds earlier. For example, if we present to the system an object A , and hold it still for a few seconds, the images generated will initially be very blurry, but the model will quickly learn to produce sharp images of A . If we then present an object B , the images generated will again initially be very blurry, but the model will quickly learn to produce sharp images of B . If we then present object A again, the images generated will be blurry again, as the model will have forgotten about A and will need to relearn.

In other words, the network is constantly *over-fitting* to the live feed, and not learning to generalize.

To overcome this, we introduce an *exponentially decaying memory*. This is a fixed size buffer of length N (e.g. $N := 2048$). With every update, we store the current image \mathbf{x}_{live} into a *random* location in the memory buffer, overwriting the current contents at that location. During training, instead of using a batch size of 1 and training on only the live image \mathbf{x}_{live} , we use a batch size of K (e.g. $K=2-4$) where one element of the batch is \mathbf{x}_{live} , and the remaining elements are sampled *randomly* from the memory buffer. We feed this batch of K images to the model to produce a batch of K output images, which we use to calculate the loss and gradients.

Using this approach, the model is always being trained on the live video feed, as well as a random selection of older images. And due to the random-in-random-out nature of the memory

buffer, the system has a higher probability of being exposed to recent images, and older images are more likely to be eventually over-written.

Finally, we include options to augment the images during training, by taking random crops and/or applying random brightness and contrast adjustments to them before they are fed into the model. While we observe this to improve generalization, for aesthetic reasons, we decide not to use this as it unsurprisingly introduces too much ‘jumping around’ in the random samples.

4.5 Experiments and results

We will now share some of the experiments that we conducted. These can be grouped into two categories which reflect the primary modes of interaction. These are *hyperparameter manipulations*, and *video feed manipulation*. We will begin by sharing our experiments in *hyperparameter manipulation*.

4.5.1 Optimiser and associated hyperparameters

The optimisation algorithm determines how the parameter space is navigated with regards to the gradients from the loss function. We implemented a number of optimisers as listed above, and we experiment with them, switching between them, and playing with their respective hyperparameters in realtime, and observing the results.

Explosions and oscillations

Two important phenomena to discuss, are *explosions* and *oscillations*.

Explosions occur when the optimiser completely shoots past a minima, and escapes the valley, into some unknown territory in the parameter landscape. The model is no where near a solution. Furthermore, the optimiser is unable to bring it back, and the system is generally unstable and out of control. This is typically when the gradient updates being applied are way too large.

Oscillations occur when the gradient updates are still large enough that the optimiser shoots past a minima, and cannot converge. However, the updates are not large enough for the optimiser to escape the valley and *explode*. Instead, the optimiser oscillates around the minima, back and forth inside the valley, without ever reaching the minima.

The visual manifestation of these can be seen in Fig. 4.5, and we will explain this image in more detail shortly.

Learning rate

A very simple way of making the optimiser *explode*, is by increasing the **learning rate** hyperparameter. This increases the *step size* that the optimiser takes with each gradient update. As a result, increasing the learning rate will first result in *oscillations*, as the optimiser consistently shoots past the minima and then returns, oscillating around the minima like a pendulum. As learning rate is increased further, the optimiser is likely to escape the valley into uncontrollable territories, and it will eventually *explode*.

Momentum

A tightly coupled hyperparameter, is *momentum*. This introduces, as the name implies, *momentum* to the equation. This is typically visualised as a ball rolling around on a hilly landscape. Higher values for momentum, equates to the optimiser preserving more of its velocity from the previous gradient update. This helps *smooth* the trajectory in parameter space, and can aid escaping local minima. This can be thought of as a *heavy* ball such as a cannonball, rolling around on the hilly landscape, unaffected by wind resistance. Whereas lower values of momentum can be thought of as a ping pong ball, losing much of its energy to wind resistance, rolling around on the same hilly landscape.

Increasing momentum can dramatically increase the step size if the optimiser is already moving in the direction of the gradient, as the velocity from the previous updates accumulate. This is analogous to a ball *accelerating* down the slope of a hill.

For this reason, *increasing momentum* is also a very simple way to make the optimiser explode. Generally speaking, it is beneficial to have momentum, as this will smooth the trajectory, and can help the optimiser escape from local minima. However, *increasing* momentum will need to be paired with *decreasing* learning rate, in order to balance out the accumulations of velocity across updates, and to prevent explosions.

Stable oscillation points

One of the interesting experiments that we tried, is to adjust learning rate and momentum in such a fine balance, that the system starts oscillating. It is stable, however, it is right at the edge of stability.

Without a Realtime Continuous interface that provides immediate feedback, such a configuration of values would be very difficult to find. However, we are able to find such configurations very quickly, within a few seconds in fact. We map our hyperparameters to faders on a midi controller, and with one finger on each fader, we do not need to take our eyes off the screen. We can slide our respective fingers forwards and backwards, and manipulate both hyperparameters simultaneously, to find the balance of values such that the system *approaches, but does not pass*, this tipping point into exploding.

There will of course be many such stable oscillating points, high learning rate and low momentum, vs low learning rate and high momentum, and everywhere in between. And each configuration will behave slightly differently. Through the relative balance of these hyperparameters, we can control the *amplitude* and *speed* of the oscillation.

The impact of novel inputs

Having found a stable oscillating point, we then introduce novel inputs to the video feed. This is typically a large, sudden movement in front of the camera, such as waving an arm. A novel input such as this, will introduce a large loss. And a large loss will introduce a large gradient. For this reason, if the system was already oscillating, in a stable manner but close to the edge of the valley, this large gradient that comes from the sudden movement is likely to cause the system to escape the valley, and ultimately explode.

Furthermore, here the differences in behaviour between the different optimisers become very noticeable.

Stochastic Gradient Descent (SGD), unsurprisingly, is the most likely to explode in a situation like this. This is because the step sizes in SGD are not dynamic. They are directly related and *fixed*, to the gradient, learning rate and momentum. If the optimiser is already oscillating at the edge of tipping while the video feed is relatively still, and the gradients are *small*, then it is very expected that it will explode if the gradients suddenly *grow*.

Optimisers such as RMSProp, AdaGrad, and Adam on the other hand, implement adaptive measures to counteract such incidents. Adam in particular, we observed to be not only the most stable, but also the quickest to converge. Building on RMSProp and AdaGrad, the Adam optimiser uses exponential moving averages to estimate the first and second moments of the gradient, to dynamically adjust both the learning rate and momentum, per parameter (D. Kingma & Ba, 2014). For this reason, even if we increase learning rate, and make sudden big movements in front of the camera, the Adam optimiser is likely to counteract massive gradients, and try to prevent large oscillations or explosions. And on the flip side, when movements in front of the camera are very subtle, and the gradients are very small, the Adam optimiser is likely to counteract by increasing the step size, and thus converges quicker.

Interestingly, and perhaps unsurprisingly, even though SGD was the *most unstable* of the optimisers, it was also the *easiest* for us to play with and meaningfully control. We were very quickly able to build a qualitative understanding of the behaviours of learning rate and momentum with SGD. We were also able to very quickly bring the SGD optimiser to an oscillating stable point. And upon exploding, we were able to bring it back under control, and back to convergence. Probably due to the non-linear complexity of the Adam optimiser, we were not able to develop as much control over the Adam-specific decay hyperparameters β_1 and β_2 .

Gradient clipping

Two more hyperparameters which are key to controlling this optimisation process, are the ***gradient clipping*** hyperparameters. These rescale the gradient if the gradient gets larger than a certain threshold, and can be a very computationally cheap and effective method for preventing explosions, or controlling oscillations (Pascanu et al., 2013). We implement two methods of gradient clipping, *L2 norm* (i.e. rescaling the gradient if its L2 norm is greater than the threshold), and *per component* (i.e. clipping each component of the gradient independently if it is larger than the threshold). We found it difficult to qualitatively assess the differences in behaviour between clipping per component or L2 norm. However, we know that theoretically, per-component clipping is likely to change the *direction* of the gradient, and this is not a desirable behaviour. For this reason, we generally use L2 norm clipping.

An optimiser experimentation session

Fig. 4.5 shows a selection of frames from an experimentation session where we play with the optimisation hyperparameters learning rate, momentum, and gradient clipping threshold, in a manner that we describe above. We have these three hyperparameters mapped to faders on a

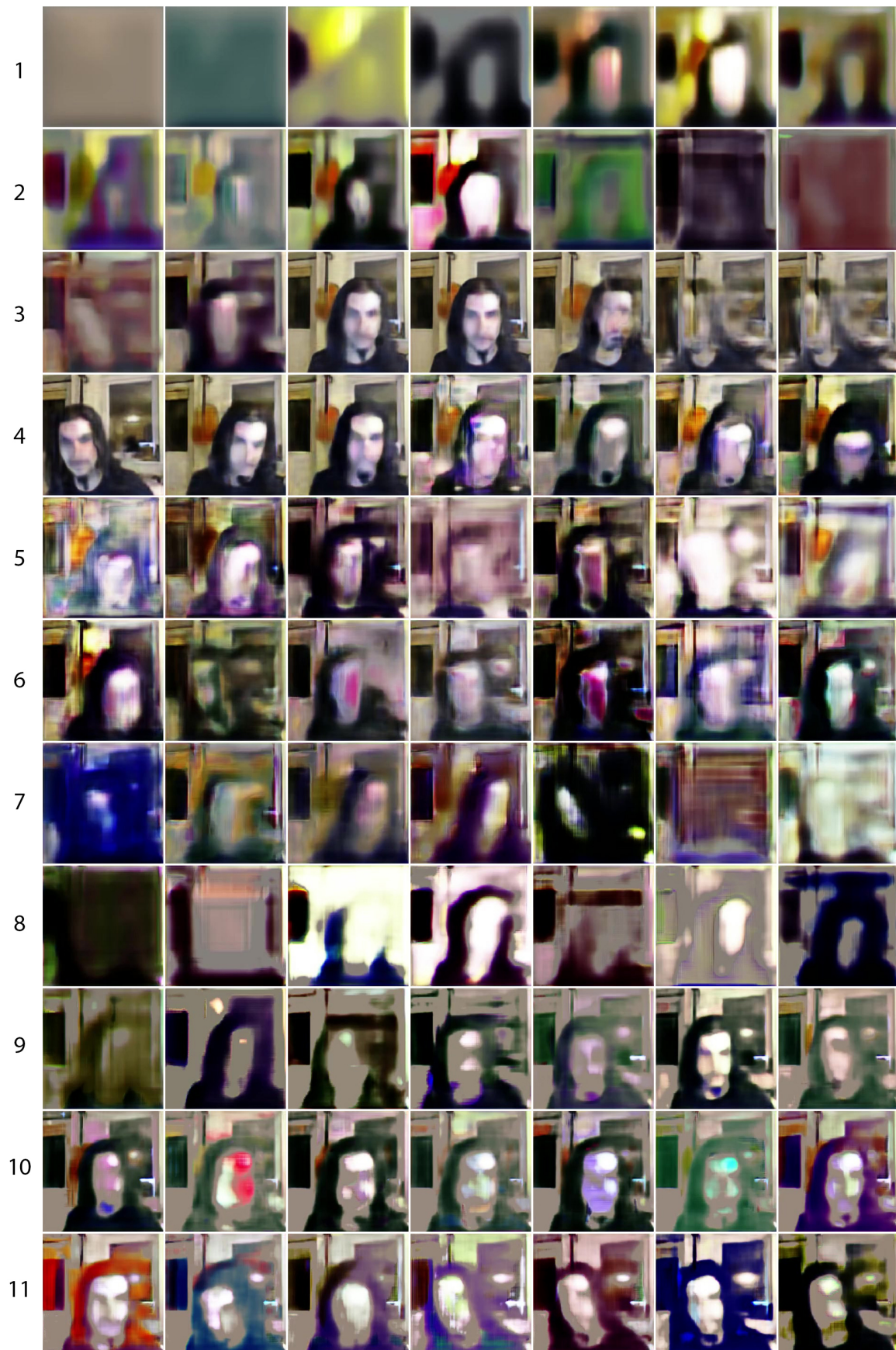


Figure 4.5: A selection of frames covering roughly a two minute experimentation session where we play with the optimisation hyperparameters learning rate, momentum, and gradient clipping threshold. Time flows from left to right, and each frame shows the live reconstruction, i.e. the *Seeing* panel.

hardware midi controller, so that we do not need to take our eyes off the screen, and we can manipulate all three hyperparameters simultaneously.

The loss function that we use is L2, and the optimiser is SGD with momentum. The live video feed is a webcam focused on our face and shoulders, similar (but not identical) to the setup in Fig. 4.4.

At the start of the session, the model is initialised randomly with Xavier initialisation (Glorot & Bengio, 2010), learning rate and momentum are set to ‘reasonable’ values 0.0003 and 0.5 respectively. The first two rows show the model converging, and by the middle of the 3rd row, the model has converged, reconstructing the video feed accurately⁵.

Towards the end of the 3rd row, we decrease learning rate to zero. This effectively pauses any gradient updates and the model stops learning. As a result, when we move to the left of the screen, the model is unable to reconstruct these novel inputs, and our head disappears. By the start of the 4th row, we restore the learning rate, and the model is able to reconstruct our head in this new location on screen.

In rows 4 to 6, we increase learning rate and momentum, until the system is oscillating wildly. This is difficult to observe in this image where the frames are spaced roughly 1-2 seconds apart. In reality, at every update, the output is wildly different to the previous in terms of colours and brightness, but some aspect of the composition is somehow always preserved. We can easily observe that the generated image is *oscillating* around an image that would be the image from the live video feed. The output flashes alternately brighter and darker, or between opposing colours, while it tries but fails to converge on the desired image from the camera.

Towards the end of row 7 and beginning of row 8, we push momentum further causing the system to explode, and the structure of the generated image is entirely lost. At every update, the output is flashing crazy colours and shapes at this point.

Towards the end of row 8 and beginning of row 9, we introduce gradient clipping to slow the system down and bring it back under control. We lower momentum and learning rate to reasonable values, and in the last 2 rows, we can see that the system is stable again, however it is oscillating. By controlling the gradient clipping, momentum and learning rate, we can control the amplitude and speed of the oscillation.

It is worth pointing out, that we do not perform any kind of colour correction, brightness, contrast or any other kinds of post processing on the images generated. These images are the raw, unprocessed outputs from the model. The model is trying to reconstruct its input as accurately as possible. The colour variations, extreme saturation, over and under exposure of the images generated, are a result of the optimiser overshooting desired minima, and entering remote territories in parameter space. Having Realtime Continuous Control over this system via a number of faders on a midi controller, we are able to meaningful control the system, and *steer* it towards desired directions, and in effect, we can play it like a visual instrument.

⁵We are using L2 loss, and for this reason the results are blurry. We will discuss this in more detail in the next section.

4.5.2 Reconstruction loss

As we discussed in subsection 2.3.17: *Variational Auto-Encoders (VAE)*, the loss function of a VAE consists of two terms, a *reconstruction loss* which ensures that the output \mathbf{x}' of the VAE resembles the input \mathbf{x} , and a *latent loss* which ensures that the latent distribution converges to a standard normal distribution $\mathcal{N}(0, 1)$.

The *reconstruction loss* is some function which returns a similarity metric between two images. Typically in ML and DL applications, functions such as L1, L2 or Cross Entropy (CE) are common, and this is known to produce blurry outputs when used with images (I. Goodfellow et al., 2014; Theis et al., 2016; Larsen et al., 2016; Dumoulin et al., 2017).

In addition to the three loss functions mentioned above, we also implement *Multi-Scale Structural Similarity Index Measure* (MS-SSIM) (Z. Wang et al., 2004). MS-SSIM is a perceptually motivated metric that compares patches of local neighbourhoods on multiple scales, and is very commonly used in image processing applications such as image and video compression. It is also differentiable, which makes it suitable for use within our gradient based optimisation process. However, MS-SSIM is very rare in the DL literature. In the case that it does appear within DL research, it is generally used for the *evaluation* and *validation* of DNN generated images. For example, measuring a deep generative model's diversity (Odena et al., 2017) or evaluating the results of up-scaling or image restoration models (Ledig et al., 2017; Dong et al., 2016; Lim et al., 2017; Johnson et al., 2016; Liang et al., 2017). MS-SSIM is generally not used as a loss function, even though it has been shown that it is superior to the de facto L2 loss, for DNN based image processing applications (H. Zhao et al., 2015). We are not entirely sure why MS-SSIM is not used as a loss function. One possible explanation might be that perhaps a majority of DL researchers prefer to investigate mathematically and theoretically robust solutions that can apply across multiple domains, as opposed to incorporating very domain-specific, perceptually motivated models. Though this is just a speculation. In our case however, since we are focusing on images in this section, we are happy to use metrics optimised for images.

A qualitative comparison of using these four loss functions can be seen in Fig. 4.6, although it is more noticeable in the accompanying video. This image shows six screenshots captured during the training of four different models. All four models have identical architecture and hyperparameters, with the only difference being the loss functions used. The L2 loss, one of the most common loss functions in Machine Learning, clearly performs the worst and produces very blurry reconstructions of the input. CE loss performs almost equivalent to L2 in terms of final *quality*. However, it converges *slower* than L2, as can be seen in the first three screenshots. L1 loss performs noticeably better than both, converging quicker and settling on a higher quality reconstruction. However, it still produces blurry images. MS-SSIM performs significantly better than all three. This is not only in terms of final image *quality*, producing the sharpest image which we can see in the bottom right screenshot. But it also converges the *quickest*, producing a recognizable image within a few seconds, as can be seen in the first screenshot.

Interestingly, we can also see that the models first learn about the *luminance* of pixels, and only once the models are able to predict luminance patterns, do they start to learn about colour. This is not a behaviour that we have explicitly programmed. It is entirely an emergent property of the Neural Network optimisation. The speed at which the four different models learn about

colour, follows the same patterns as we describe above. By iteration 293, colour has just started to appear in the MS-SIM model while the other models are still monochromatic. The L1 model follows shortly behind, and by iteration 502, has full colour while L2 and CE are still mostly monochromatic.

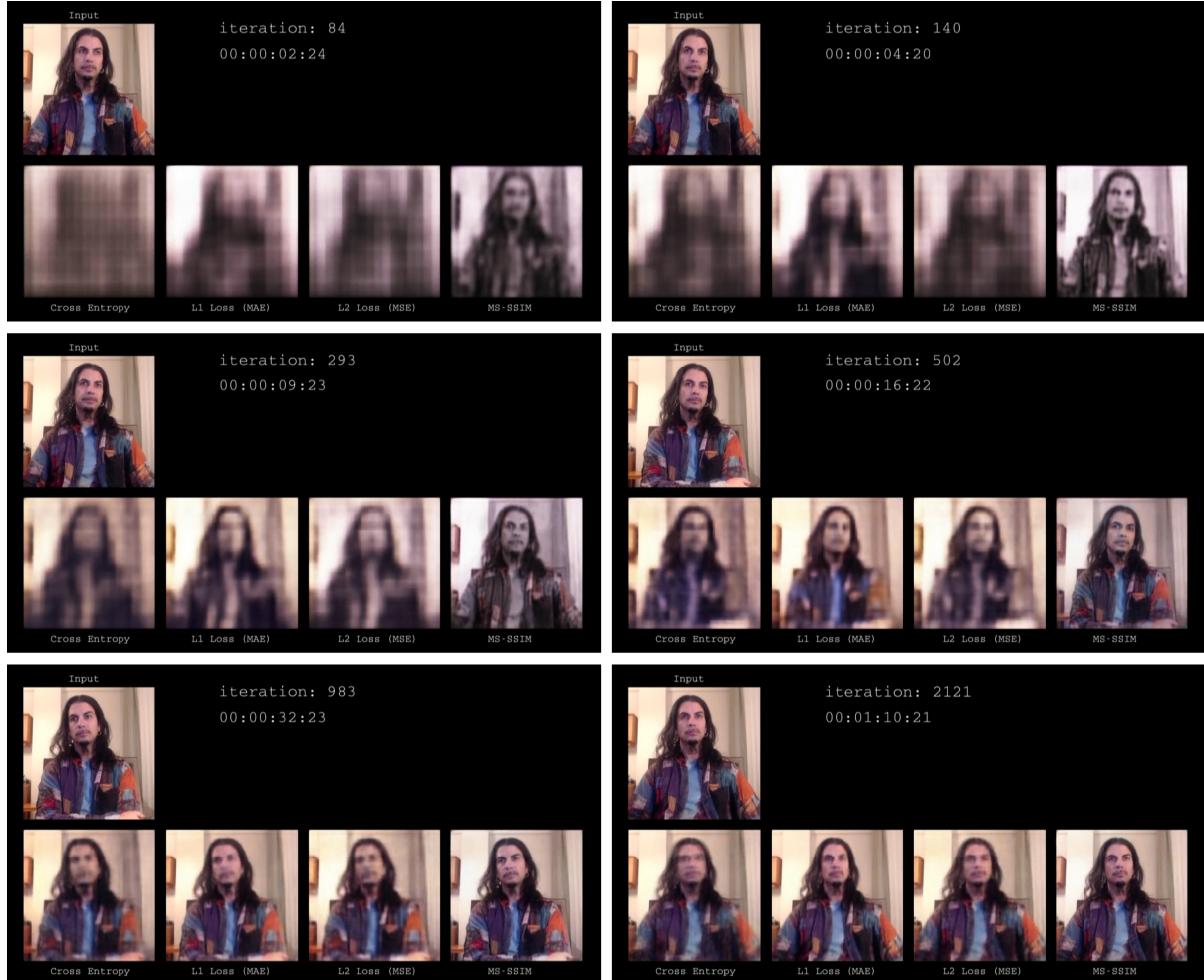


Figure 4.6: Six screenshots showing a qualitative comparison of the four different loss functions. Four models train in parallel, with identical architecture and hyperparameters and the only difference being the loss functions. MS-SSIM converges the quickest, producing the sharpest reconstructions, while L2 and CE loss produce the most blurry.

4.5.3 Latent loss and variational reparametrisation

The second term of the VAE loss, is the *latent loss*, which consists of the KL divergence between a standard normal $\mathcal{N}(0, 1)$ and the latent distribution. This ensures that the latent distribution converges to a standard normal.

As we have briefly touched upon above, we noticed that the performance of the VAE depends heavily on the relative weights between the reconstruction loss, and the latent loss. In other words, there is a decision and compromise that can be made, whether we prefer the VAE to produce more accurate *reconstructions* (\mathbf{y}_{live}), or more believable *random samples* ($\mathbf{y}_{\text{random}}$). This can be controlled through changing the balance of the loss terms. Prioritizing latent loss, leads to poorer reconstructions. However, the latent space is more structured and disentangled,

and random samples are of higher quality. Prioritizing reconstruction loss, leads to higher quality reconstructions. However, the latent space is less structured, and potentially sparse. It may not be distributed in a standard normal distribution as we desire. Thus, random samples are of a lower quality and do not always accurately resemble the training examples.

Furthermore, this balance is already dependent on and biased by the dimensionality of the input and the latent layers. This is to say that, the reconstruction loss is already weighted by the input layer’s dimensionality K , as seen in eqn. (2.12). And the latent loss is already weighted by the latent layer’s dimensionality J , as seen in eqn. (2.13).

As a result of this, if we decide for example to increase the dimensionality of the latent layer, in order to observe the impact on the model’s *expressive power*⁶, we will not in fact be observing the effects of an isolated change. This is because, not only has the model’s architecture changed as a result of this, but the loss function has changed dramatically as well, the latent loss is weighted more heavily in comparison to before.

To address both of these issues, we implement the solution that we discuss in subsection 2.3.17: *Variational Auto-Encoders (VAE)*. We normalise both the reconstruction loss and latent loss, and we multiply the latent loss by a new hyperparameter W_{KL} .

This W_{KL} hyperparameter, is an additional hyperparameter that we can manipulate in realtime. Through realtime experimentation, we found that a value in the range 0.01–0.1 provides a subjectively satisfactory compromise, with higher values prioritizing higher quality random samples, and lower values prioritizing higher quality reconstructions. We also verified this through thorough offline traditional hyperparameter search.

We also implement a pathway in the computation graph that bypasses the variational reparametrisation altogether. This effectively toggles between a *Variational Auto-Encoder*, and a *normal Auto-Encoder*. Disabling the variational component and switching to a normal AE results in the model ceasing to be a *generative* model. In other words, it no longer captures the distribution of the data. The model is still able to *reconstruct* the live video feed. However, the distribution of its latent layer is no longer a standard normal. In fact, in this case we do not have any knowledge as to how the latent layer is distributed. Furthermore, this distribution is not structured such that we can exploit the properties that we discuss in subsection 2.1.3: *Latent manipulations*.

In the following section subsection 4.5.4: *Video feed manipulations*, we will discuss our experiments in *video feed manipulations*. And we will share our observations with regards to the *Reminiscing* panel, which, as we briefly explained in section 4.4: *System description*, is a *random sample from the local neighbourhood* of the live video feed. As a result, *in a well structured space of latent representations*, we would expect the image in the *Reminiscing* panel, to be somehow similar to the live feed.

In a non-Variational, normal Auto-Encoder, this is not the case. The images may be distributed anywhere in the unbounded, infinite space of the bottleneck layer. Only in the *Variational Auto-Encoder* do we observe this.

⁶subsection 2.3.7: *Universal function approximators, expressive power*.

4.5.4 Video feed manipulations

So far we have only discussed realtime manipulation of *hyperparameters*. However, there is an additional mode of interaction which we have not yet discussed. That is the act of simply moving or manipulating objects in front of the camera.

In this section, to save space on the pages, we tightly crop screenshots around the *Stimulus*, *Seeing* and *Reminiscing* panels, as we will be focusing on these. In all of these example, we are using MS-SSIM loss, and the Adam optimiser. For this reason, the reconstructions are much sharper than our previous experimentations. We are also using a batch size of 2 for training, where the second image is randomly selected from the exponentially decaying memory buffer. This ensures that the model doesn't 'forget' immediately, and doesn't over-fit to the live video feed.

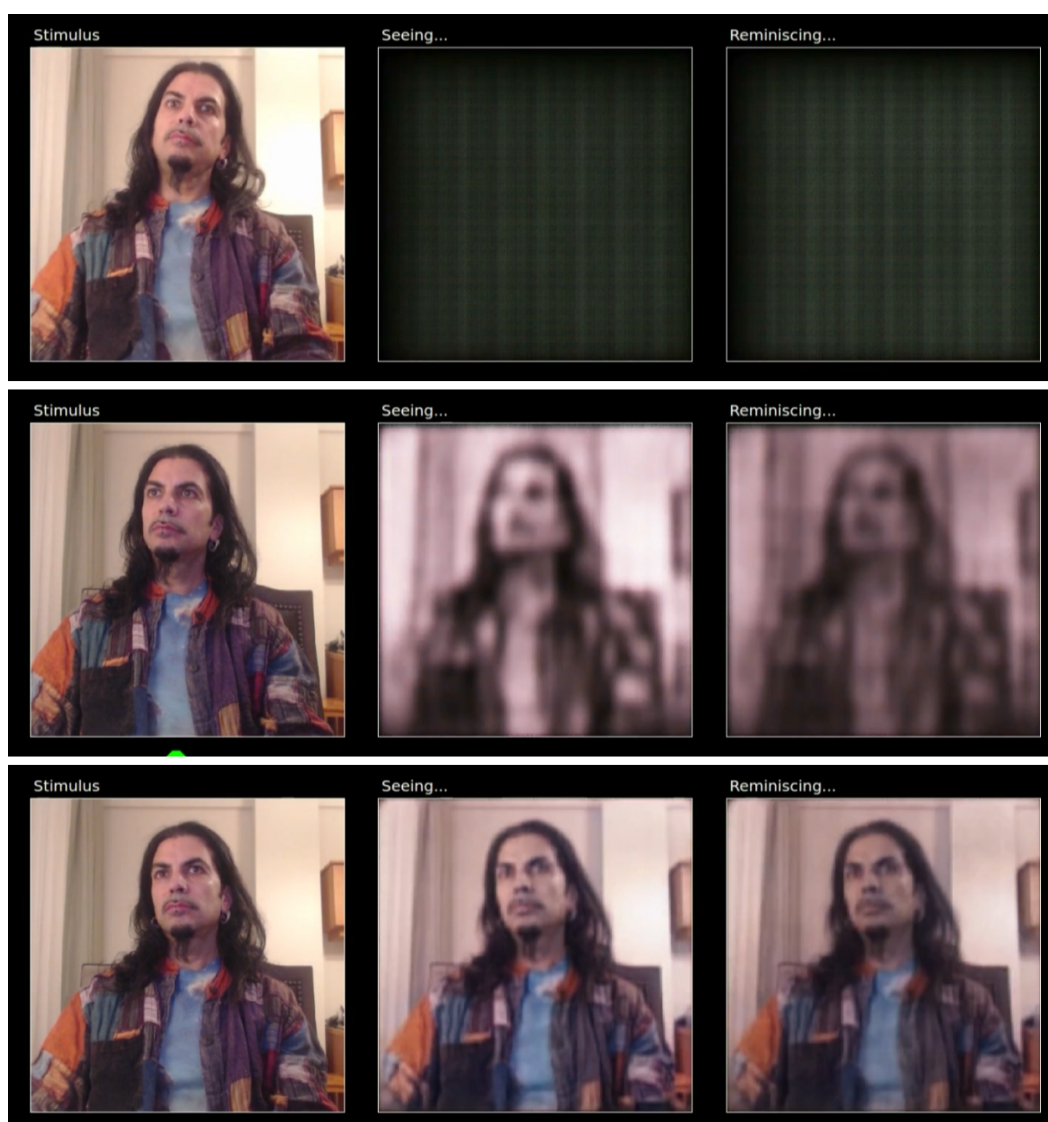


Figure 4.7: Three screenshots taken during the first 30 seconds of a new experimentation session. The model is initialised at the beginning, and by the 3rd row, has converged to the extent that the reconstructed image *Seeing* is sharp and resembles the live video feed *Stimulus*. This indicates that the model has become familiar with, and has learnt the necessary features required to represent and reconstruct this *Stimulus* image.

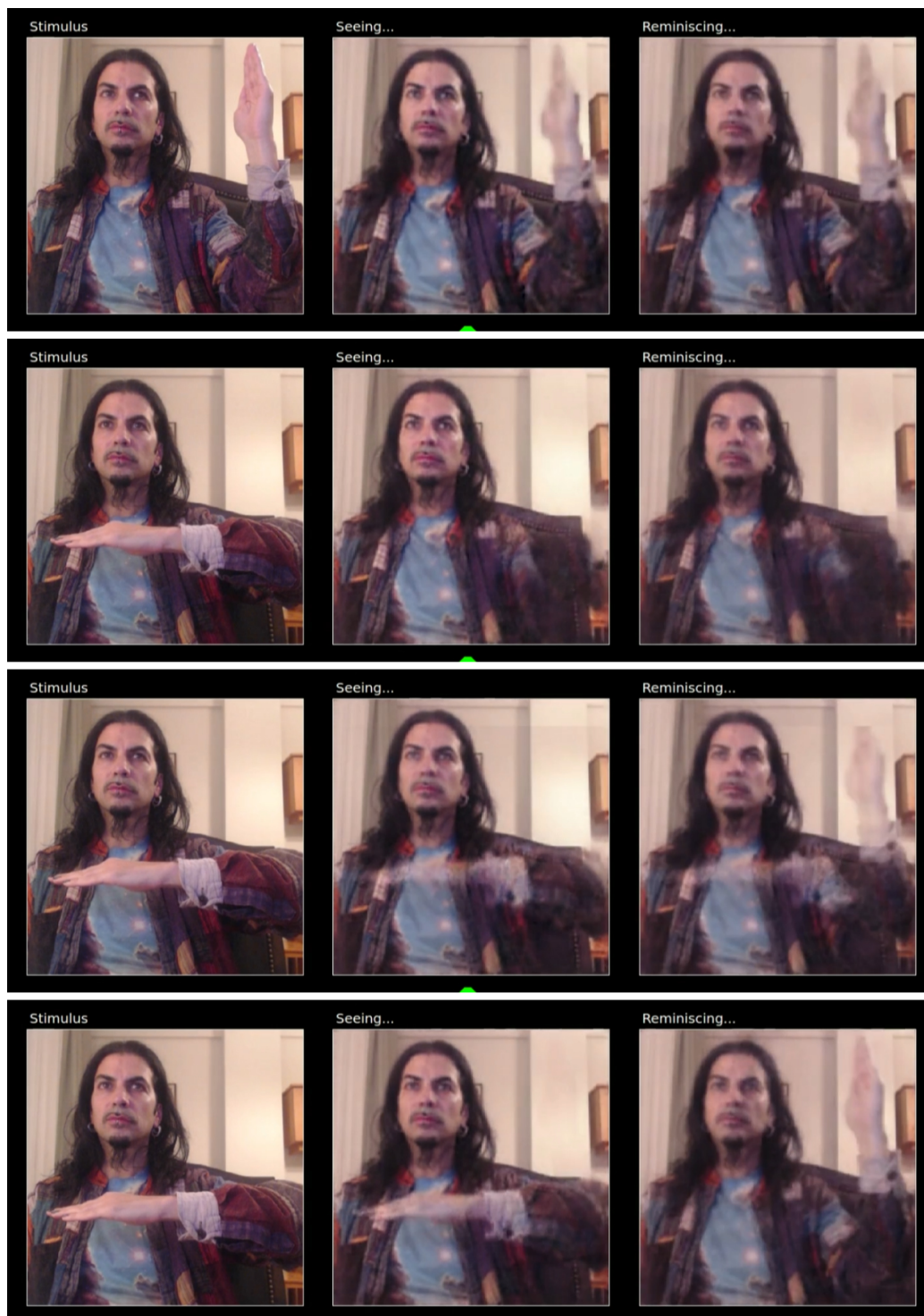


Figure 4.8: In the top row, we hold our left arm vertical and hold it still for a few seconds while we wait for the reconstructed image *Seeing* to become sharp. We then rotate our arm to be horizontal. As can be seen in the 2nd row, the entire reconstructed image *Seeing* is sharp, except for our arm, which is missing. The model has not seen our arm in a horizontal position yet, and thus lacks the representations required to visualise it in that manner. Across the last two rows, the arm flickers and fades in as the model learns the necessary representations for the arm in this position. Note that in the 3rd column *Reminiscing*, the arm flickers between horizontal and vertical. This is due to the fact that the *Reminiscing* image is a random sample from the local neighbourhood of the *Stimulus*. And in the well structured latent space of a generative model, this local neighbourhood will contain images that are aesthetically, structurally, and/or semantically similar. For this reason, the *Reminiscing* image will be random samples that *resemble* the *Stimulus*, without necessarily being straight reconstructions.

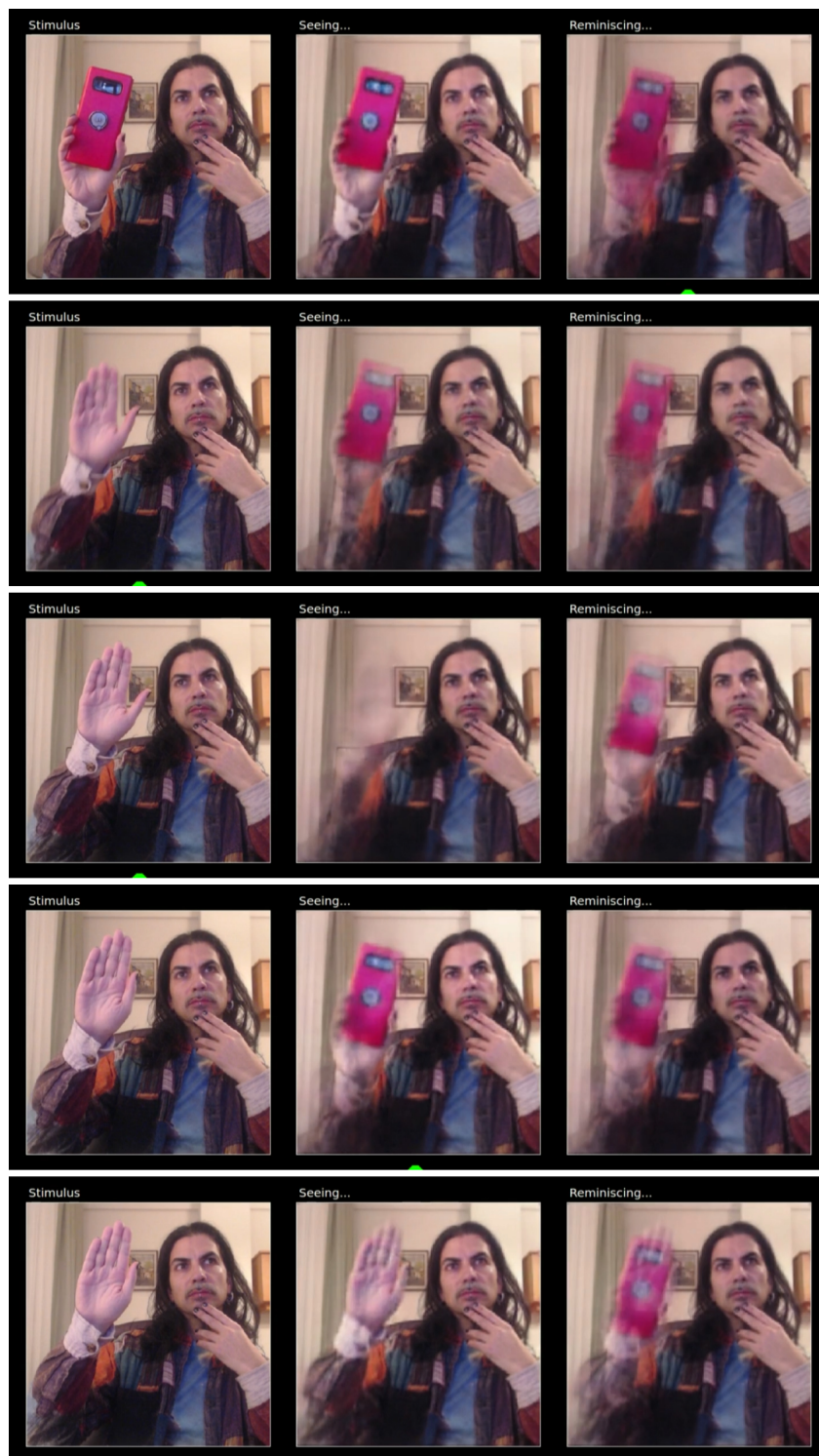


Figure 4.9: In the top row, we hold a red phone in our right hand for a few seconds while we wait for the reconstructed image *Seeing* to become sharp. In the second row, we put the phone down, and quickly lift our hand to the same position. In *Seeing*, trying to reconstruct this image of our hand without a phone, the model samples the *closest image* it can to this *Stimulus*. And that is limited by what it has seen before, and the representations that it has learnt. In this case, the red phone appears in our hand. Over the course of the subsequent rows, the phone flashes on and off and gradually fades out as the model learns the necessary representations for this *Stimulus*. In the *Reminiscing* panel however, we can see that the local neighbourhood of this point in latent space is still dominated by the red phone. It will take another minute or two for the distribution to be evened out across this larger area in latent space.

4.5. EXPERIMENTS AND RESULTS

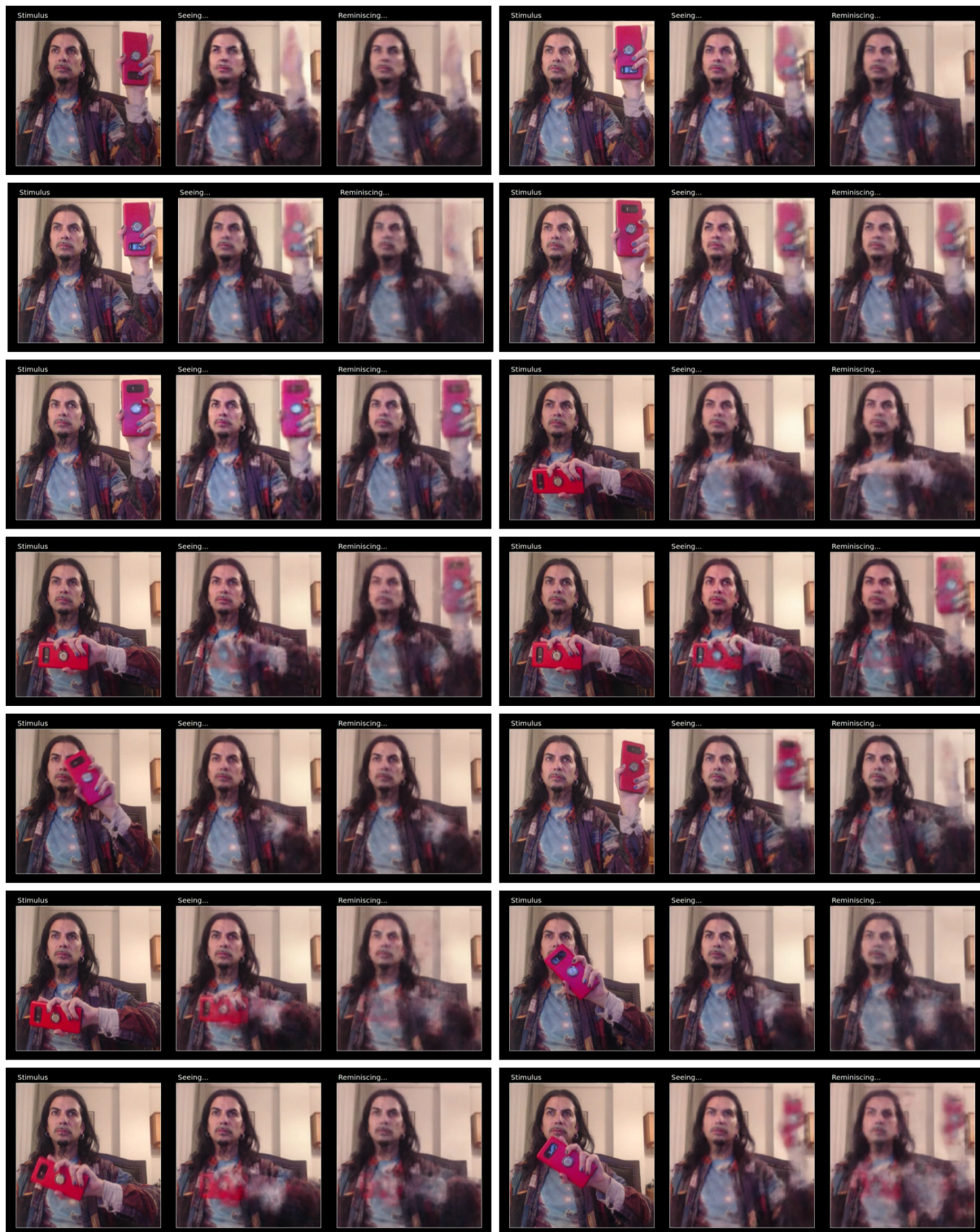


Figure 4.10: We hold the red phone in our left hand. However the model has not seen this. In fact it has only seen our arm in this position with our hand open and flat, as it was in Fig. 4.8 . So in *Seeing*, this is how the model reconstructs this *Stimulus* image. Over the next few frames, the red phone gradually fades in, flickering in and out. After the 5th frame, we start slowly rotating our arm back and forth between horizontal and vertical, and we observe the results in both *Seeing*, and *Reminiscing* as the phone flickers in and out, and the arm even snaps to horizontal or vertical positions.

4.6 Conclusions

In this chapter, we present a system that trains in realtime on a live video feed. While the system is training, we are able to manipulate a number of hyperparameters in realtime via a GUI, and immediately observe the results.

Manipulating for example W_{KL} , a weight multiplier on the latent loss term, we were able to immediately observe the results. With this realtime control and immediate feedback, we were able to identify a range for W_{KL} that we found provided a desirable balance between reconstruction quality, and random sample quality. We were able to use this range as a starting point for a traditional offline hyperparameter search, which we confirmed as 0.01–0.1. In this case $W_{KL} := 0.01$ provides sharper reconstructions at the cost of lower quality random samples, and $W_{KL} := 0.1$ provides blurrier reconstructions but higher quality random samples. We discuss this in more detail in subsection 4.5.3: *Latent loss and variational reparametrisation*.

Manipulating the reconstruction loss function in realtime also proved to be very useful. We saw that our MS-SSIM converged much quicker than L1, L2 and Cross Entropy. It also provided much sharper images. L2 and Cross Entropy performed the worst, both in terms of final image quality, and speed of convergence. We provide more details and images for comparison in subsection 4.5.2: *Reconstruction loss*.

Perhaps the most useful hyperparameters to manipulate in realtime were the optimisation parameters. These include learning rate, momentum, gradient clipping thresholds, and optimiser function. Unsurprisingly, we found that the more advanced *Adam Optimiser*, was quickest to converge, and was also the most robust in terms stability. The simplest optimiser *SGD with momentum*, was a lot more unstable. However, because of the simplicity of the momentum-based SGD optimisation algorithm, we found that it was the easiest to learn how to control.

And this is where perhaps the biggest surprise of this particular study came. Our initial aim with this study, was to observe how DNNs train in realtime, and hopefully build a qualitative understanding of the hyperparameters. We realised however, that the system that we had developed had incredible performative potential. We were able to build a qualitative understanding of the optimisation parameters such that we could play with them in realtime, and control the outputs generated by the Neural Network in a meaningful manner. We were able to bring the system into oscillatory states, bordering between stable and unstable. We were able to push the system to explode, and bring it back under control again. We discuss all of these experiments in subsection 4.5.1: *Optimiser and associated hyperparameters*.

The most important finding of our experience with this system, was that we were able to generate images with the system that we did not know the system was able to create. And we were able to explore this space in a Realtime Continuous manner, with Meaningful Human Control, and use the system as a performative instrument.

In the next chapter, we build upon the system that we developed in this study, and we investigate ways of providing more control over the aesthetics.

Chapter 5

Learning to see: Digital puppetry through realtime video transformation

5.1 Introduction

In the previous chapter chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, we presented a method and software that we developed, that trains a Deep Neural Network in realtime on a live camera feed. Using this software allowed us to gain very valuable insights into how Artificial Neural Networks learn. Using that insight, we were able to control the images generated by the Neural Network, in a *realtime, meaningful and continuous* manner. The system that we developed made this possible via two primary modes of interaction. The first mode of interaction is a very direct control of the *input* to the system. By manipulating objects in front of a camera, a user can control the images fed to the Neural Network for processing. This provides a very *performative* and *playful* interaction. The second mode of interaction, is through realtime manipulation of a number of parameters via a Graphical User Interface (GUI) or a physical device such as faders on a midi controller. We found that these two modes of interaction, complement each other very well, and provide a wide range of meaningful control to the user. In this chapter, we wish to investigate this further.

One of the aims of the research that we presented in *Hello World*, was to investigate *how DNNs learn*. And in particular, we were able to adjust a large number of hyperparameters in realtime, while a DNN was training¹. With immediate feedback, we were able to observe the effects of the various different hyperparameter configurations, and this allowed us to optimally select values inline with the behaviours that we were looking for.

In this chapter, we move beyond that question, and we wish to investigate how we can provide *more control over the aesthetics* of the images generated by a Deep Neural Network. We expand on the software tool that we developed in the previous chapter, and we build upon what we consider to be a very powerful aspect of the system, which are the two modes of interaction we describe above. In *Hello World*, we provide no additional training data to the system. Instead,

¹See subsection 2.3.13: *Hyperparameter search* for a discussion on hyperparameters and hyperparameter search.

upon launch of the software, the Artificial Neural Network starts blank, and trains live on a camera feed. In this chapter, we introduce to this system large datasets consisting of thousands of images. In other words, we first *pre-train Neural Networks* on these large datasets, and then we investigate *how can we use these pre-trained models within our system*, and provide *realtime, meaningful, continuous control to the user*, via the *modes of interaction* that we describe above.

It is worth mentioning, that the system we developed, and will present in this chapter, is capable of both *starting with a model pre-trained* on a dataset of thousands of images, while *continuing to train live* on the camera feed (and/or another dataset of images) as it did in *Hello World*. While this provides very interesting results, we found that it distracts from the main question that we seek to address in this chapter, which is to *meaningfully control the aesthetics* of the images generated by the Neural Network. Enabling realtime learning, on top of pre-trained models, makes it much more difficult for a user to isolate and understand the creative impact of any actions that they might take, as the system is far more complex, with orders of magnitude more variables affecting the images generated. This in turn, makes the control available to the user, less *meaningful* in accordance with the working definition that we present in this thesis. For this reason, for all of the work that we present in this chapter, we choose to disable realtime learning, and we leave that for future research which focuses solely on the question of *how do Neural Networks learn*. Instead, we focus only on integrating large datasets into our realtime, interactive system.

In this chapter, under the umbrella title *Learning to see*, we will present and discuss a number of related outputs. These include a publicly released open-source software tool, a number of public videos and artworks, and the underlying method that we used to create all of these works. We shared our videos and artworks publicly on the internet, and this constitutes a crucial aspect of our work. This is because, as we discuss in chapter 1: *Introduction & Motivations*, our primary aim is not necessarily to invent specific methods that accomplish certain tasks. It is instead, to *demonstrate the potential power* of realtime, meaningful, continuous control over the outputs of generative Deep Neural Networks. This field, at the time of our research in 2016–2017, is practically non-existent. We hope to help *encourage more research and discourse* in this direction. Many of the works that we created, gained immense popularity via the internet, and we consider this to be an indication that realtime, meaningful, continuous control over the outputs of generative Deep Neural Networks is indeed a very important area of research. One of the video demonstrations that we created, was shown during Nvidia CEO Jensen Huang’s keynote at GTC (GPU Technology Conference) 2019², with the voiceover “[AI is] inventing new ways to bring out the creative genius in us all”. We presented the research at SIGGRAPH 2019 (Akten et al., 2019). The work has been included in MIT’s Open Documentary Lab³, and many of the videos have been shared widely on social media and influential mainstream blogs and publications such as gizmodo⁴, boingboing⁵ and prostheticknowledge⁶. The works

²Nvidia GTC 2019 Keynote: <https://www.youtube.com/watch?v=Z2XINfCtwI&t=32>

³<https://docubase.mit.edu/project/learning-to-see/>

⁴<https://gizmodo.com/trippy-magic-happens-when-ai-only-knows-about-flowers-1823900244>

⁵<https://boingboing.net/2018/03/21/watch-neural-networks-see-only.html>

⁶<https://prostheticknowledge.tumblr.com/post/172012841001/gloomy-sunday-latest-addition-to-memo-aktens>

have also been shown in galleries, museums and cultural institutions such as The Barbican (London, UK), Ars Electronica Centre (Linz, AT), Moscow Museum of Modern Art (Moscow, RU), Itaú Cultural (Sao Paulo, BR), International Documentary Film Festival Amsterdam / IDFA (Amsterdam, NL) and many more.

We have also released a simplified version of our software as open-source software on github (Akten, 2017). This version contains a subset of the filters that we mention in this chapter, and lacks many of the advanced non-DL related options such as advanced screen layout options, changing and loading models at run-time, midi controller support etc. In simplifying the functionality and code, we hope that it will be more informative and educational, and can act as a base for others to build upon in the future. The github repository currently has 311 stars and 61 forks.

5.2 Overview

The key question that we seek to address in this chapter, is the exact question at the core of this thesis, applied to images: How can we design and develop *interactive generative systems* that exploit and leverage the *capabilities of state-of-the-art Deep Learning algorithms*, while allowing *Meaningful Human Control* over the generated images, in a *Realtime Continuous* manner?

When we train a generative Deep Neural Network on a large dataset of images — for example, of fire — the trained model is able to generate an infinite number of images of fire. However, instead of generating *random* images of fire, we wish to design a system in which a user is able to control the generated images in a meaningful way. If the user envisages a particular *shape* of flames, or fire of a particular *intensity*, or embers placed in particular layouts etc., they should be able to easily produce the types of images that they desire. Furthermore, the interaction with the system should be in realtime, and continuous. This would allow the user to experiment, explore, search and find configurations that produce desirable outcomes.

We present a method in this chapter, which tackles this question. The method that we present, is not specific to fire, or any specific domain of images. It is entirely generic, and automatically works with any categories of images, without the need for domain-specific customization. Furthermore, our method doesn't only operate on images, but on live video feeds, and thus we can produce videos in realtime, through performative and playful interaction.

At the heart of all of the works that we present in this chapter, is a *realtime, interactive, video processing and transformation system*. We utilize a highly customizable and parametrised image processing pipeline, combined with a generative Deep Neural Network which we train with a novel training and data augmentation system. This system that we've designed, is not specific to flames, or flowers, waves, images of space or any other of the examples that we present in this chapter. Instead, our system is very generic, and *learns* and exploits any types of regularities or diversity that may exist in a dataset.

We think of our system as a *visual instrument* that allows users to *create, conduct* and *play with* visual media, with realtime, expressive and playful interaction. Our system processes a live *video feed* — either from a camera or a video file — and reconstructs new images that resemble the input image in composition and overall shape and structure, but is comprised of

the aesthetic characteristics determined by a large corpus of data. At first glance, this may sound similar to *Neural Style Transfer*⁷. However, our system uses a very different approach, in that while Style Transfer typically extracts ‘style’ information from one or a few dozen images, we train our models on *thousands* of images. As a result, our models contain information from across a much larger domain. When combined with a parametrised image processing pipeline, this grants a user many opportunities for meaningfully controlling the images generated in a number of different ways. Building on this system, we provide three modes of interaction:

Digital puppetry Directly controlling the contents of the video feed is one of the more performative and playful approaches that our system allows. By manipulating our hands, body or any kinds of objects in front of a camera, we can control the overall composition and structure of the generated images. We see this as a kind of *digital puppetry*. Examples of this can be seen in Fig. 5.1 and Fig. 5.2 and we discuss it in more depth in subsection 5.4.2: *Digital puppetry*.

Augmented drawing In addition to *puppeteering* objects in front of a camera, we can also control the contents of the video feed by filming ourselves *draw* or *paint* on a piece of paper or any other surface. Examples of this can be seen in Fig. 5.3 and we discuss it in more depth in subsection 5.4.1: *Augmented drawing*.

Live parameter manipulation A very powerful feature of our system, is that it has a number of *human-understandable* parameters that allow a user to control various characteristics of the output image in a meaningful way. For example, these include parameters to produce *brighter* or *darker* images. But instead of just brightening or darkening an image using a post-processing filter as Adobe Photoshop might do, our system is able to produce images which are naturally brighter or darker, by using appropriate *features*. This can be seen in Fig. 5.4 where our system gradually introduces clouds of nebulae to reconstruct an input image with brighter features.

These parameters can be adjusted in realtime via a Graphical User Interface (GUI), or a physical interface such as a hardware midi controller with dials and faders. We discuss these parameters in much more detail in subsection 5.4.3: *Live parameter manipulation*.

Providing a realtime, immediate feedback loop between the user’s actions and the system’s outputs, enables the user to experiment, improvise and *perform* the visual media. No laborious keyframe placements, no animation curve adjustments, no rendering options are necessary; everything is performed and captured live.

⁷We discuss this in more detail in section 2.2.7: *Neural Style Transfer (2015)*

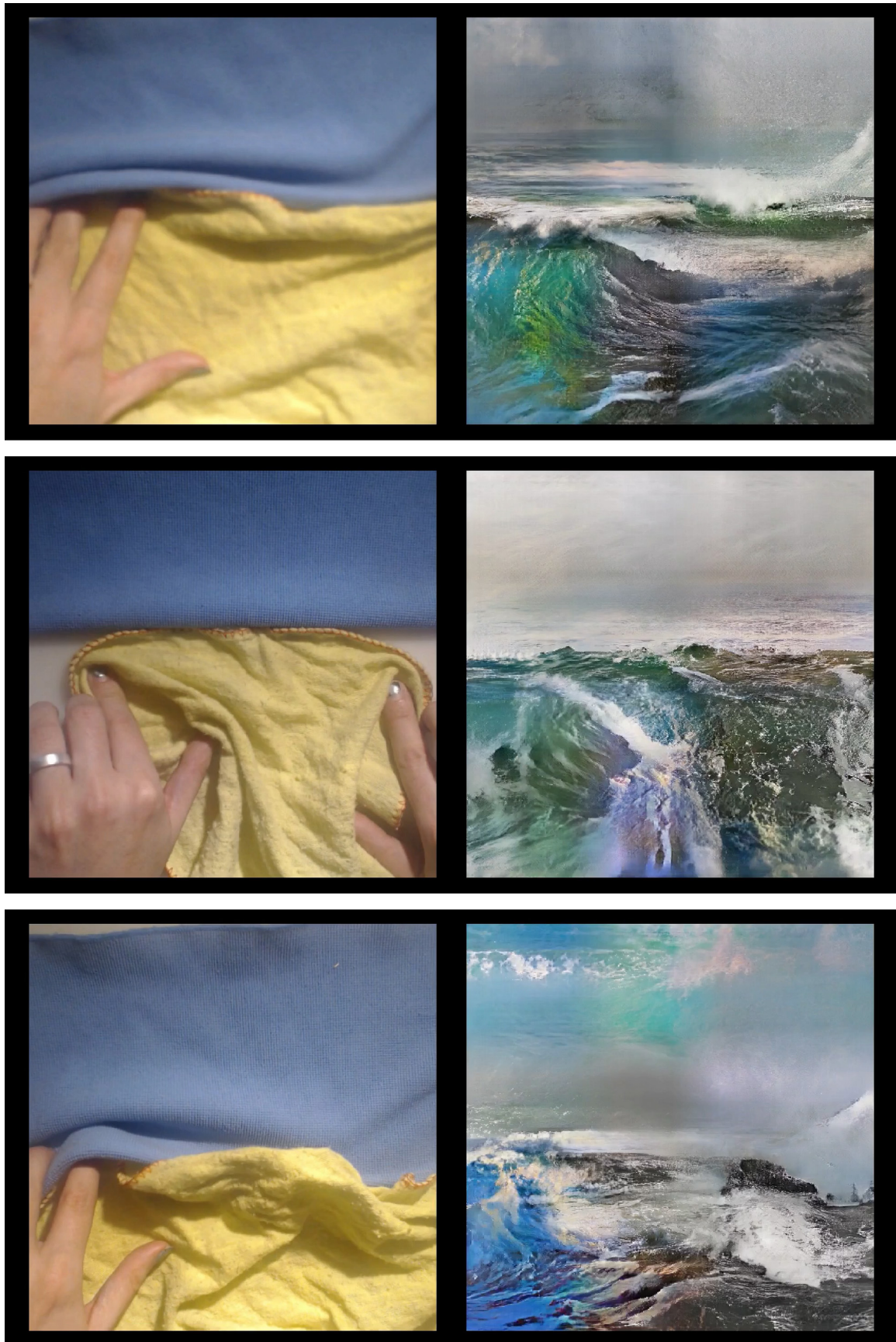


Figure 5.1: Frames from *Gloomy Sunday* (video, 2017), demonstrating subsection 5.4.2: *Digital puppetry*. Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.



Figure 5.2: Frames from *Gloomy Sunday* (video, 2017), demonstrating subsection 5.4.2: *Digital puppetry*. Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.

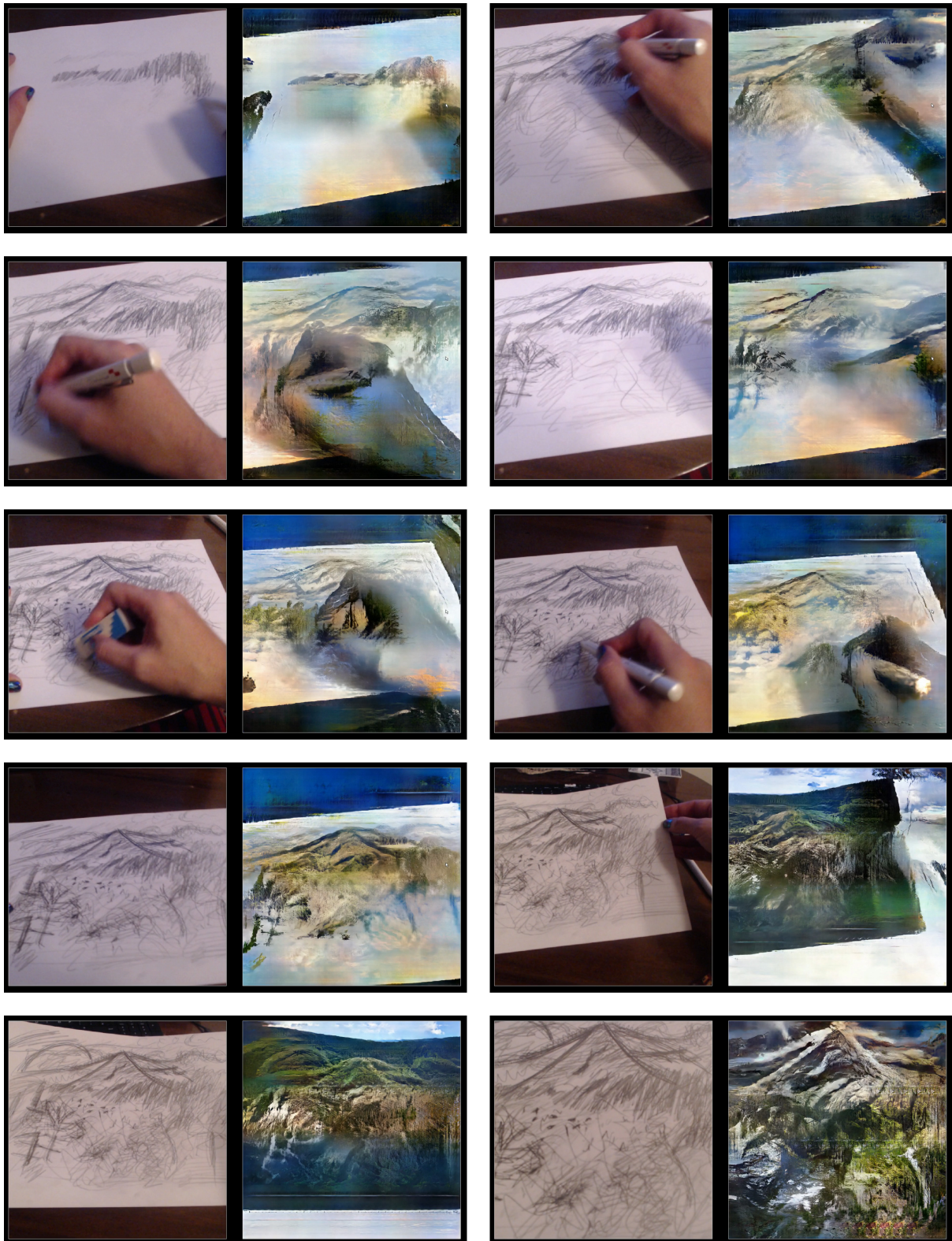


Figure 5.3: Frames from a video demonstrating subsection 5.4.1: *Augmented drawing*. Each pair of images shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image generated in realtime by our software.

5.2. OVERVIEW

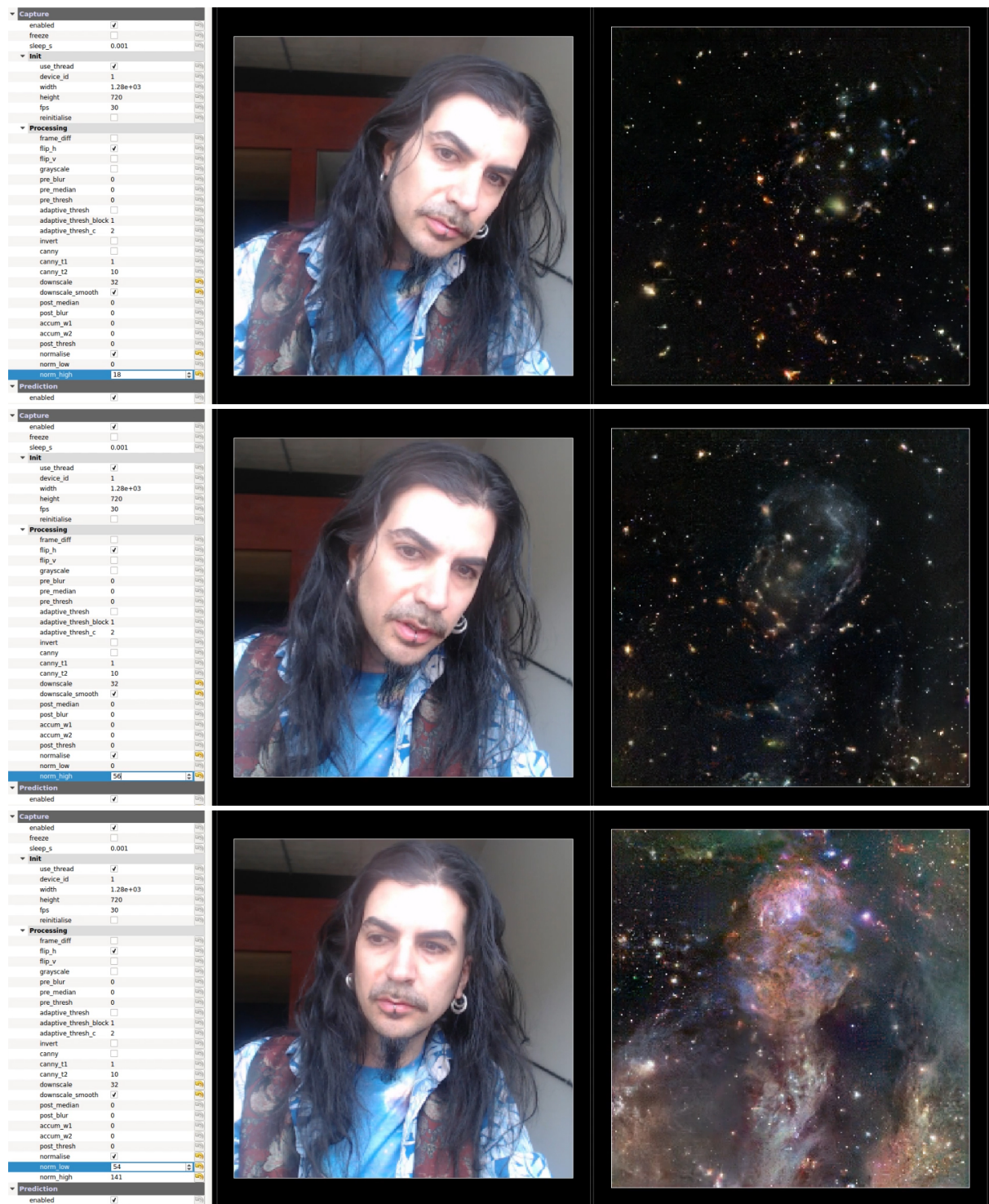


Figure 5.4: Frames from *We are all made of star dust* (video, 2017), demonstrating subsection 5.4.3: *Live parameter manipulation*. In the top frame, we adjust the parameters such that the desired output brightness is low. The system automatically uses dark features to construct the output image. In this case, most of the output image is comprised of distant galaxies, while hints of larger and brighter distant galaxies define the outline of the face in the video feed. In the lower two frames, we increase the desired brightness, and the system automatically replaces the distant galaxies with brighter features, such as nebulae. We demonstrate this in much more detail in later sections.

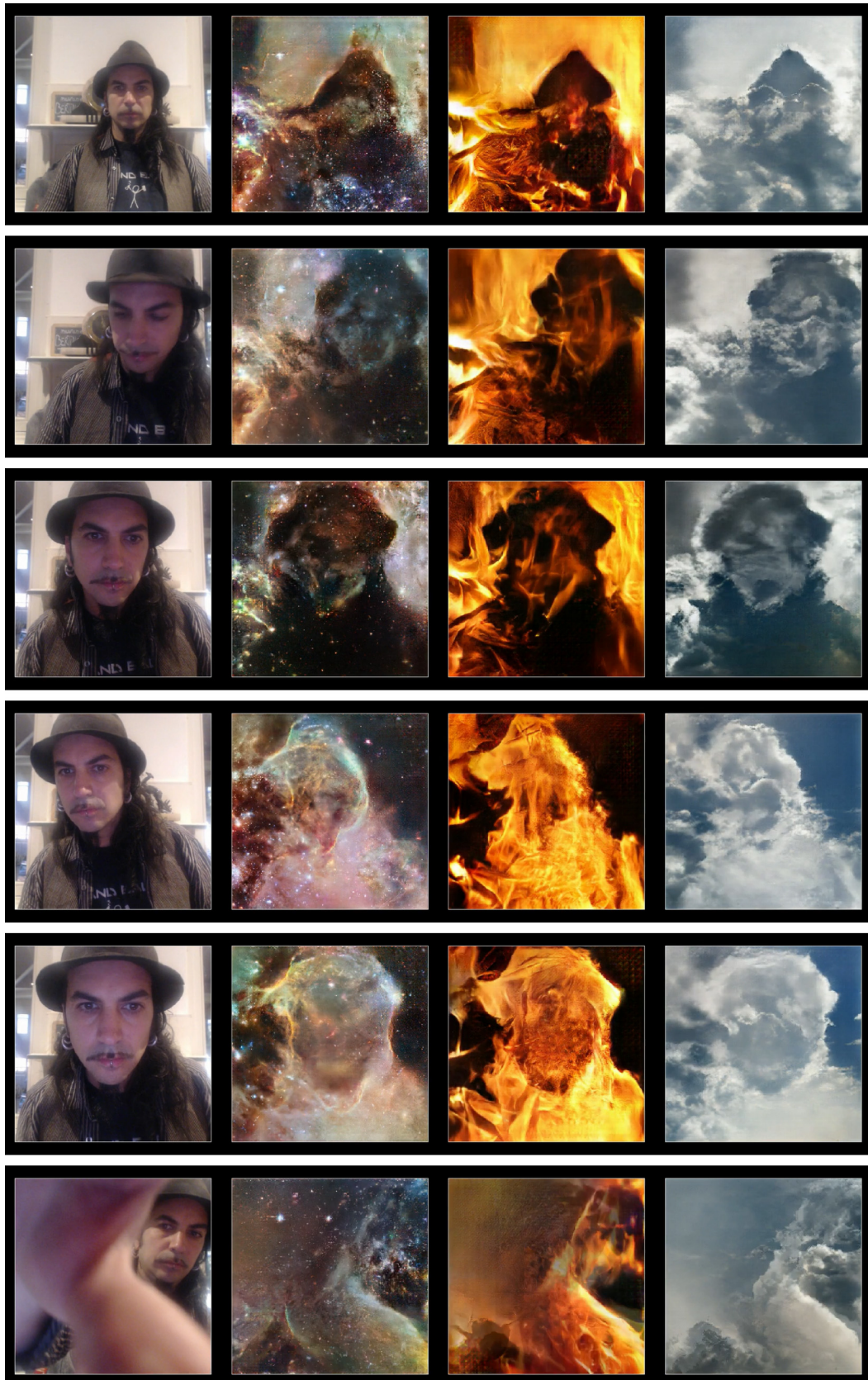


Figure 5.5: Frames from a video demonstrating *multiple simultaneous models*. This was presented as an *interactive installation* at *International Documentary Film Festival Amsterdam (IDFA) 2017*. Each row shows a single frame from the video, and is also a screenshot from our software (with the GUI hidden). In each frame, the left panel shows the live video feed from a camera, and the subsequent panels show images generated in real-time by our software using multiple models trained on different datasets.



Figure 5.6: section 5.4.2: *Interactive installation* as exhibited at The Barbican's 2019 *AI: More than Human* exhibition in London, UK and currently on tour around the world. We discuss this in more detail in section 5.4.2: *Interactive installation*.

5.3 System description

Before we present more of our results, and demonstrate the full capabilities of our system in more detail, we will first discuss some of the key technical implementation details. This will allow us in later sections, to discuss our results in context of this technical information.

As we previously mentioned, in *Learning to see*, we do not use realtime learning. Instead, we first train a number of models on large datasets of images, in a separate training phase. Once the models are trained, we load them into our *realtime inference software*. This software feeds each frame from a live video feed – either from a camera or a video file — to the selected model(s), and displays the results in realtime. While this is happening, the user can also manipulate a number of *human-understandable parameters* in realtime via a GUI, or a physical interface such as a midi controller, to influence the output in a meaningful and expressive manner.

The Neural Network architecture we use is based on *pix2pix* image-to-image translation (Isola et al., 2016)⁸. We add additional layers to provide up to 512x512 pixel resolution. At the time of our research in *Learning to see*, this architecture provided state of the art results in high resolution image-to-image translation. However a year later, researchers from Nvidia and UC Berkeley introduced a novel architecture allowing up to 2048x1024 pixel resolution image-to-image translation, colloquially known as *pix2pixHD* (T. C. Wang et al., 2018). The method and pipeline that we discuss below, can also be adapted to *pix2pixHD*. However, due to the increased complexity of the *pix2pixHD* architecture, the training times are drastically longer, and inference performance falls below interactive rates on our hardware (Nvidia GeForce 1080 Ti): 5fps at 512x512 with *pix2pixHD*, vs 15fps with our modified 512x512 *pix2pix*. Since our emphasis is on *realtime*, playful, expressive interaction, the results that we show in this thesis are from our original 2017 *pix2pix* models.

On top of the image-to-image translation network, there are two additional key aspects that we introduce, that allow our system to perform in the way that it does. The first of these, is the way in which we augment the data during training. The second, is a multi-stage parametrisable image processing pipeline. A very high level schematic of the system can be seen in Fig. 5.7. We will discuss these in more detail in the following sections.

5.3.1 Datasets

Our models are trained on datasets consisting of thousands of images that share a desired aesthetic or content type. In the examples that we present in this chapter, we will focus on five different models. These models are *Waves*, *Fire*, *Clouds*, *Flowers* and *Deep space*. We collected the data for these models by batch downloading images from the photo sharing website *Flickr* that were tagged with the respective categories. For the *Deep space* model, we also downloaded many images from the Hubble Space Telescope’s website.

In addition to the motivations that we outlined above, and inline with the premise of this thesis, we also have additional, higher level conceptual motivations behind selecting these particular categories of images. However, as this is not relevant to the *technical implementation details*, we will leave that discussion for section 5.5: *Conclusion*.

⁸Please see subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)* for details.

5.3.2 Training

The pix2pix architecture is a conditional GAN, which we discuss in more detail in subsection 2.1.2: *Conditional generative models*. This model is designed for *paired image-to-image translation*. The generator network learns a mapping $G : \{\mathit{input}, \mathit{z}\} \mapsto \mathit{target}$, where z is a random noise vector within the latent distribution of the model, input is an input image, and target is the desired target image. For this reason, the training data for pix2pix models typically require $(\mathit{input}, \mathit{target})$ image pairs.

For the training of the models in our system, we only explicitly provide target images, and do not provide input images. Instead, we implement an image processing pipeline that constructs input images from the target images, on the fly, during training. In addition, for each of the image transformations in the pipeline, we pre-define a range, within which the parameters are randomised on every training iteration, individually for each image. A summary of this training pipeline, including a typical stack of transformations with the pre-defined ranges can be seen in Alg. 2.

The image processing pipeline can be thought of as consisting of three stages: i) *training only*, ii) *training & inference* and iii) *inference only*. Training in this manner, with the input images generated on the fly during training, and a three stage image processing pipeline, grants our system the flexibility and power that it needs in order to behave in the way that we desire.

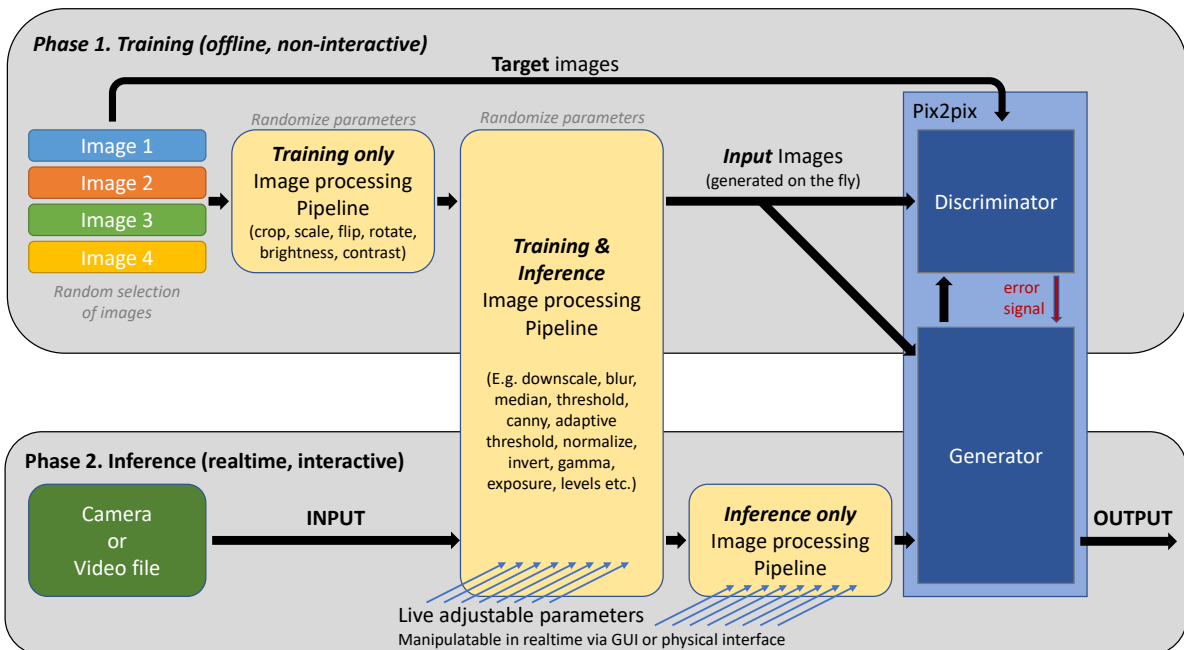


Figure 5.7: A high level schematic of the system including the training and inference stages. During pix2pix training, the images which are to be used as inputs in the $(\mathit{input}, \mathit{target})$ pairs, are generated on the fly from the target images via a custom image processing pipeline. The parameters of this pipeline are randomised on every iteration, in order to augment the dataset and aid generalisation. During inference, the video feed is also fed through the same image processing pipeline. In addition, a number of additional image processing filters are applied before the images are fed into the model for prediction. This exposes a number of human-understandable parameters for realtime manipulation and meaningful control over the images generated.

Algorithm 2: Learning to see: Training pipeline

```
// Options
1 desired_size: desired image size for the model. e.g. (512x512);
2 permit_flip_horizontal: (Boolean) depends on the nature of the dataset;
3 permit_flip_vertical: (Boolean) depends on the nature of the dataset;
4 permit_rotate: (Boolean) depends on the nature of the dataset;

// Training image processing parameter 'magic values'
// Found via thorough trial and error
5 scale_min: 100%;
6 scale_max: 130%;
7 downscale_amount: 24;
8 brightness_amount: 20%;
9 contrast_amount: 15%;

10 foreach training_iteration do
11     begin prepare targets
12         targets ← load random selection of images for this mini-batch;
           // Process each target image with different random parameters
13         foreach target in targets do
14             uniformly scale target to Random(scale_min, scale_max) * desired_size;
15             take random crop of desired_size from target;
16             if permit_flip_horizontal and RandomBool() then flip horizontally;
17             if permit_flip_vertical and RandomBool() then flip vertically;
18             if permit_rotate and RandomBool() then
19                 rotate RandomSelect(90, 180, 270) degrees;
20         end
21     end

22     begin prepare inputs
23         inputs ← convert targets to greyscale;
24         begin apply low pass filter
25             downscale inputs by downscale_amount with area filtering; // scale down
26             upscale inputs by downscale_amount with cubic filtering; // restore size
27         end
           // Process each input image with different random parameters
28         foreach input in inputs do
29             adjust brightness by Random(100-brightness_amount,
30                 100+brightness_amount);
31             adjust contrast by Random(100-contrast_amount, 100+contrast_amount);
31         end
32     end

           // Train on one mini-batch
33     Pix2pix_Train(inputs, targets);
34 end
```

Data augmentation The *training only* stage of the image processing pipeline consists of transformations such as *scale*, *crop*, *flip*, *rotate*, as well as *brightness* and *contrast* adjustments. Every time a *target* image is selected for a mini-batch during training, it is processed with these transformations, with parameters randomly sampled from within the pre-defined range. This ensures that the network will never see the same images, or even the same image *patches*, in the exact same way more than once. This encourages the network to generalize to novel inputs, instead of just memorising the training data.

Most of the datasets that we use and present in this chapter, contain images that have a sense of an absolute *up* and *down* direction, but not necessarily an absolute *left* or *right* direction. For example, images of the ocean, or fire, etc. For this reason, we randomly flip these images horizontally, but not vertically during training. This is denoted by the *permit_horizontal_flip* and *permit_vertical_flip* options in Alg. 2. Images of deep space however, have no absolute *up* or *right* directions. For this reason, we can randomly flip those images horizontally and/or vertically, and we can also randomly rotate them by 90, 180 or 270 degrees, to further augment the dataset. This is denoted by the *permit_rotate* option in Alg. 2.

It is important to remember, that each of these transformations of *scale*, *crop*, *flip*, *rotate*, *brightness* and *contrast*, are applied randomly and uniquely, to every image, at every training iteration. As a result, our model is exposed to novel inputs on each and every training iteration, and is able to generalise more effectively to the real world inputs that it will receive during inference.

Input abstraction Our models are trained on datasets consisting of thousands of images that share a desired aesthetic or content type. In our realtime inference software, when we feed these trained models images from a video feed such as a camera, or an arbitrary video file, it is very unlikely that the video feed will resemble the training data in any way. For this reason, we apply filters to the images before they go into the Neural Network, both during training and during inference. We do this in order to abstract the input images to a degree, such that they will always share some kind of basic, abstract, aesthetic. In other words, at the end of this filtering stage, *any* images that are input to the models during inference, will *always* resemble the aesthetics of the *filtered* training inputs, to some degree.

The primary filters that we have tested and used for this purpose, include *low-pass filters*, *median* and *edge detection (canny)*. Many of the examples that we present in this chapter use a low-pass filter, as we have found through many experiments, this provided the most useful and flexible results. More specifically, we downscale the input image 24x, and then upscale it to the original resolution. This acts as a heavy but cheap-to-compute *blur*. We settle on the value of 24x, after training hundreds of models, on various different datasets, trying out many values in the range 2x to 64x. We found 24x to provide a subjectively preferable balance, whereby an image input for inference can be abstracted to the extent such that the trained model can produce convincing output images, while the details of the input image are still preserved to some degree. We also desaturate the input image.

Using these particular filters on the Neural Network inputs effectively trains the model to *unblur* and *colourize* a very blurry, abstract, greyscale blobby image. The Neural Network

learns to do this, and constructs a new image with the aesthetic qualities and details of the training data, and with the overall shape, structure and composition of the inference input. In other words, during inference, the video feed is first *desaturated* and *blurred* by the image processing pipeline, and then *unblurred* and *colourized* by the Neural Network. This can be seen quite clearly in Figures 5.8 and 5.9. Fig. 5.8 shows examples of the training data — both in its original form, and the processed versions generated on the fly during training. The model tries to learn to predict the first column from the middle column, and produces the results in the third column.

It is worth mentioning, that the image processing pipeline that we describe above, is what we use for most of the examples presented in this chapter. However, for the subsection 5.4.1: *Augmented drawing* examples, we instead use *canny edge detection*. In the original pix2pix (Isola et al., 2016), the authors demonstrate a similar image-to-image translation example whereby they translate outline drawings of handbags, to photographs of handbags. For this they use *Holistically-Nested Edge Detection (HED)* (Xie & Tu, 2017) to produce outline drawings of handbags. The internet famous *edges2cats* (Hesse, 2017) also uses HED to produce outline drawings of cats. We found that HED did in fact sometimes give subjectively preferable results over canny. However, this came at a very significant speed-performance cost. In other words, when using HED instead of canny, the framerate of our realtime inference software fell drastically, compromising the realtime, expressive nature of the interaction. Furthermore, we found that using canny, paired with the ability to adjust its parameters in realtime, and within an image processing pipeline consisting of other simple filters such as median, blur, gamma etc., provided infinitely more control over the output, without compromising speed or quality. We will discuss these in much more detail in subsection 5.4.3: *Live parameter manipulation*.

Live parameter manipulation during inference The ability to manipulate the parameters of the image processing pipeline in realtime, is one of the most powerful aspects of our system. We discuss this in depth with many examples in subsection 5.4.3: *Live parameter manipulation*.

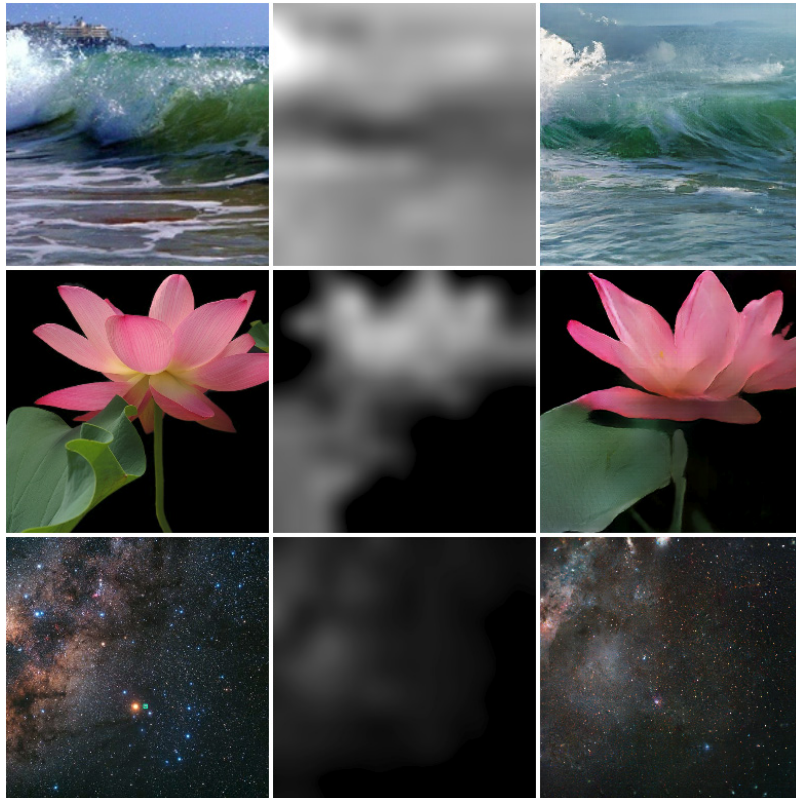


Figure 5.8: Example images from *training*. Each row shows examples from a separate model. From top to bottom, these are the *Waves*, *Flowers* and *Deep space* models. The left column shows an image selected at random from the training data. This is a random crop, and may or may not have been flipped. This will be used as a *target* image for this particular training iteration. I.e. it is the *ground truth*. The centre column shows the *input* image generated on the fly via realtime processing of the *target* image, with randomised parameters (i.e. random brightness and contrast). The right column shows a prediction from the generative model, given only *input*. In other words, the model is learning to predict the first column, given only the second column, and is producing the third column.

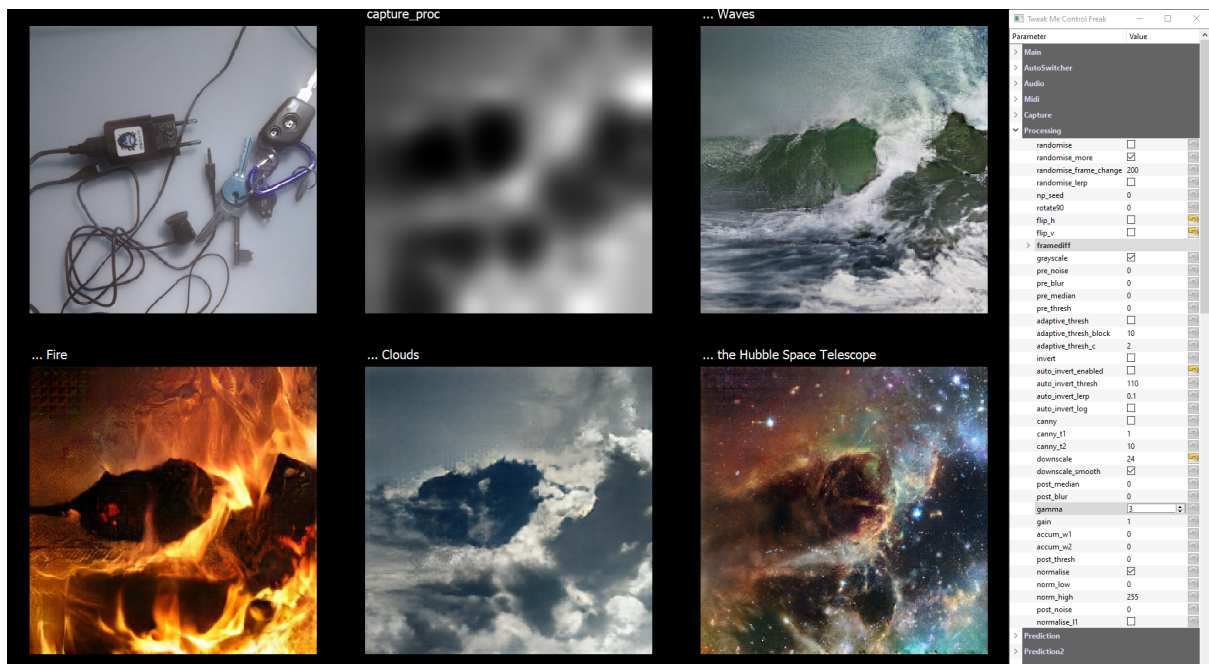


Figure 5.9: A screenshot from our real-time inference software with multiple models and the GUI enabled. In the top left corner, we can see the live video feed, either from a camera, or a video file. In the top centre, we can see the video feed processed by the image processing pipeline. This is the *input* image (labelled *capture_proc* in the screenshot) that is fed to the trained model(s) for inference. In this particular case, the video feed has been *downscaled* 24x, and then upscaled back to its original size. This acts as a low-pass filter. The next four panels show the outputs from four different models, simultaneously running inference on the same input. On the far right, we can see the GUI, showing the parameters which are adjustable in realtime. Notice how the original video feed in the top left, is radically different from the raw training examples in the left-most column of Fig. 5.8. However, the centre image, which is actually *input* into the model during inference, very much resembles the types of images in the centre column of Fig. 5.8, which are input into the model during training.

5.3.3 Inference

For this work, we have considerably built upon the real-time video processing software that we presented in the previous chapter chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*. Even though this software is still capable of training in realtime, as we mentioned in the introduction of this chapter, for the purposes of this particular research, we have disabled realtime learning. For this reason, we think of this software now as a *realtime inference software*. In this application, we have implemented many features that vastly streamline our research. Some of these relate to graphics and screen layout options. We also have the ability to select between different input video feeds at runtime. Our software supports USB cameras, network IP cameras, video files, or folders of images. We also have the ability to load and switch between trained models at runtime. Our software can process the input video feed through a single model, and cycle through multiple models via a keyboard input, or automatically on a timer. Alternatively, the software can process the same input video feed through multiple models simultaneously, and display the results side by side (Fig. 5.5), or in a grid (Fig. 5.9). However, as these particular options do not directly relate to the focus of this thesis, we will not dwell on them in any more detail.

5.4 Experiments and results

5.4.1 Augmented drawing

Videos demonstrating *augmented drawing* were among the first of our *Learning to see* outputs that we shared publicly on social media. Figures 5.10 and 5.11 show examples from 2017 and 2018 respectively, with more examples in Fig. 5.3. We also shared the source code for this version of our training and realtime inference software on the code sharing platform github⁹.



Figure 5.10: Frames from a video demonstrating *augmented drawing* (2017).

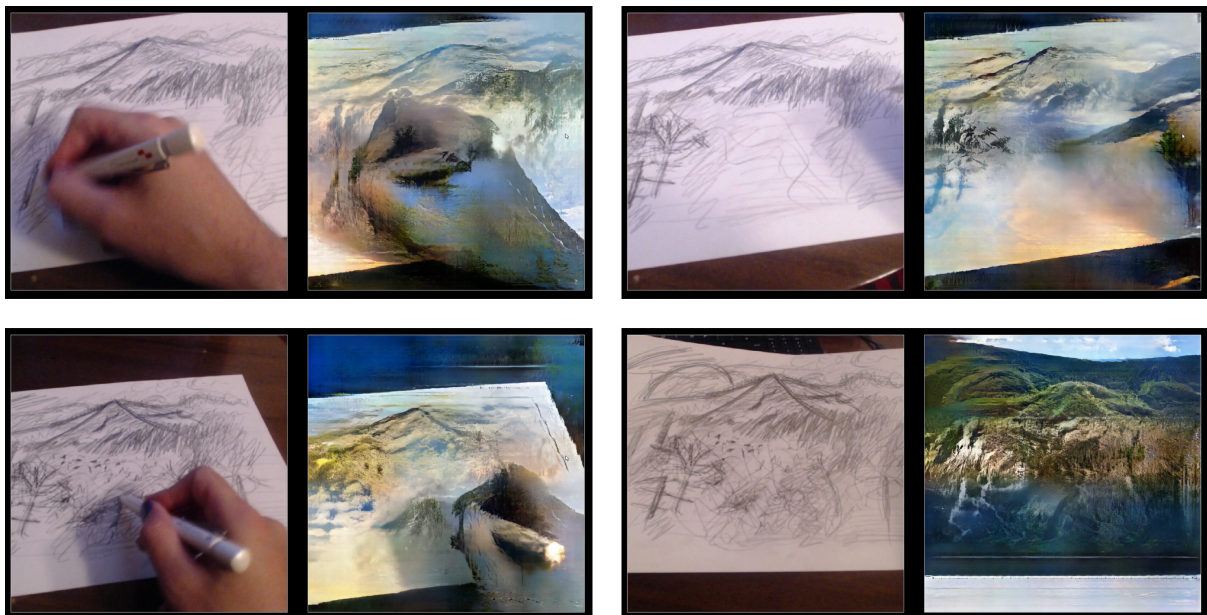


Figure 5.11: Frames from a video demonstrating *augmented drawing* (2018). More frames from this video can be seen in 5.3.

⁹<https://github.com/memo/webcam-pix2pix-tensorflow>

In this mode of interaction, a user draws simple lines on a piece of paper, and an appropriately trained model will ‘colour in-between’ the lines, or re-interpret them, with the aesthetic qualities and details of the dataset that the model was trained on. In this respect, it is similar to some of the examples presented in the original pix2pix research (Isola et al., 2016), for example *edges-to-handbags*, and the internet famous *edges2cats* (Hesse, 2017).

Our main point of focus however, is to integrate such a model, with a *live camera feed*. Most importantly, we prioritize *instantaneously realtime*, and *continuous* interaction. This provides immediately noticeable advantages compared to a *screen-based* and *turn-based* drawing interface such as the one provided in *edges2cats*. In a turn-based drawing interface, the user first makes a drawing, then they submit the drawing for processing, and then they can see the results after a second or so delay. After seeing these results, they can continue drawing to repeat this cycle. However, we believe that *instantaneously realtime*, and *continuous* interaction is essential to maximize expressivity, and we design our system with that in mind. In other words, every little action that a user makes, is instantly and continuously processed, and the results are displayed instantly and continuously.

It is possible of course, to implement instantaneously realtime and continuous interaction within a screen-based drawing interface. This would indeed improve the expressivity of the interaction. However, a live camera feed input provides additional benefits such as independence from the screen, greater analogue control, and potentially increased accessibility for people who have strong skills with pens and paper, but not computers.

Shortly after we shared our videos and code, artist and roboticist Patrick Tresset started experimenting with a similar system that he developed, quoting our work as inspiration. Tresset trained models on thousands of drawings that he had been collecting over the years, made by robots that he had built. In the videos that he produced ^{10 11 12}, it is particularly interesting to see Tresset layer many thin lines, continuously drawing over them, gradually thickening them, while he observes how his model responds to his actions, and he reacts accordingly. At times, he covers parts of his drawing with his hand, or a blank piece of paper, in order to observe how that section of his drawing influences the way that the model produces its results. In his video, Tresset explains “*I found the results interesting as a way to explore how the model works... Sketching with it is a fascinating process. It is a very good way to explore and try to understand the model. It gives ideas on how to curate/compose the dataset*”. Stills from these videos can be seen in Fig. 5.12.

¹⁰Delusions #1 <https://www.youtube.com/watch?v=9n65s88sFBo>

¹¹Delusions #2 <https://www.youtube.com/watch?v=mQNj6-PcBw>

¹²Delusions #3 <https://www.youtube.com/watch?v=ojk9KAbqt1E>



Figure 5.12: Stills from *Delusions* (video, 2017) by French artist and roboticist Patrick Tresset. Each pair of images shows a single frame from the video. In each frame, the left panel shows the live video feed from a camera, and the right panel shows the image output from his software in realtime.

5.4. EXPERIMENTS AND RESULTS

In 2019, the American artist and human anatomy expert Scott Eaton, further distilled this process, with models that he trained on thousands of photographs that he had taken of the human form over the years. In the videos that he has shared online¹³, Eaton demonstrates remarkable results with regards to how his models have learnt to light and shade the shapes that they generate. Eaton sketches simple lines and curves on a piece of paper, and his system colours in the lines with human flesh-like textures, complete with lighting, shading and shadows. Eaton then further experiments with drawing additional lines to create creases, ledges, overhangs, indents and outdents, observing how his actions are impacting the lighting and shading of the models' output. Stills from some of Eaton's videos can be seen in Fig. 5.13. Notably, the last row of images, show Eaton experimenting with crumpling and twisting the piece of paper in front of the camera, and observing the results. This provides a nice transition to the next section, in which we will discuss the mode of interaction that we call subsection 5.4.2: *Digital puppetry*.

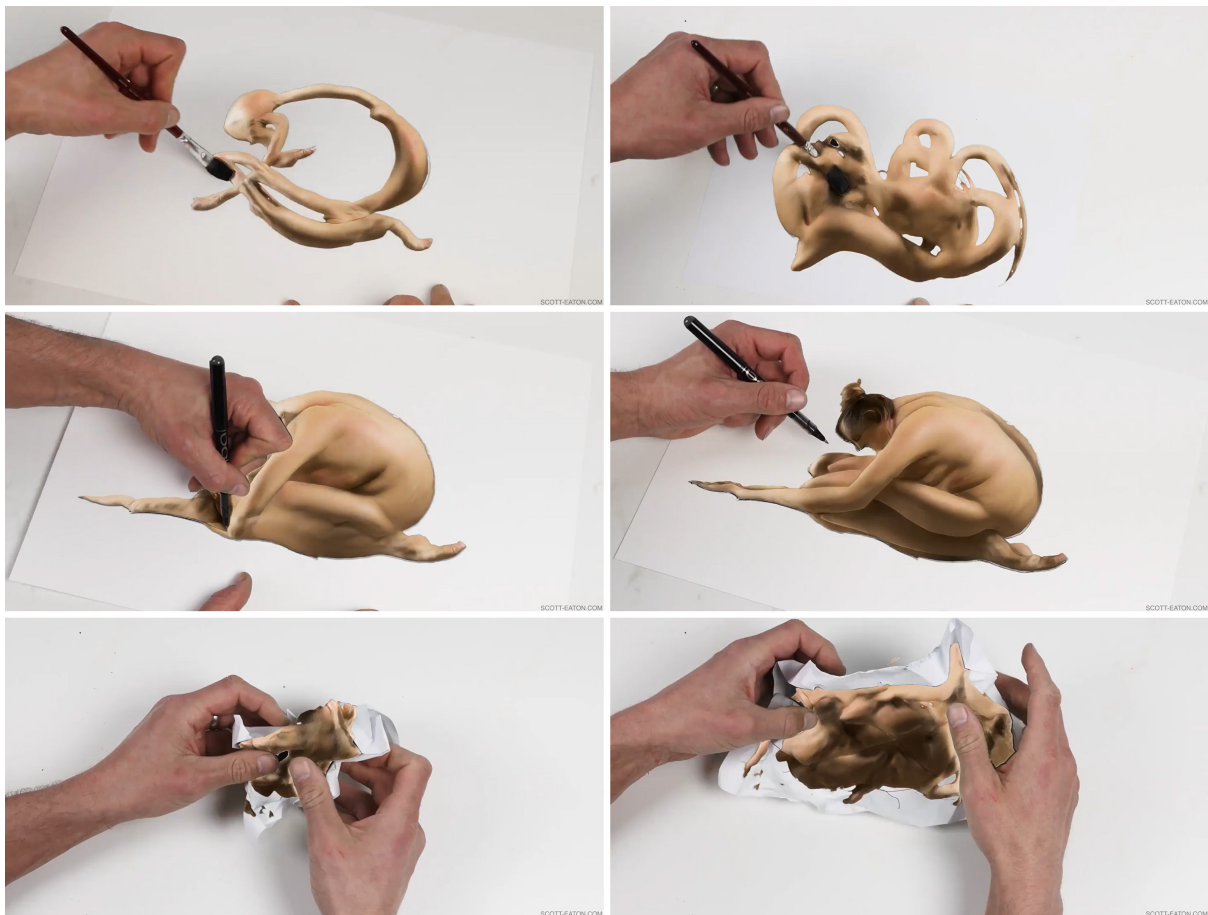


Figure 5.13: Stills from *Body Network on Paper I and II* (video, 2019) by American artist and human anatomy expert Scott Eaton.

¹³<http://www.scott-eaton.com/>

5.4.2 Digital puppetry

The mode of interaction that we call *digital puppetry*, is simply the act of manipulating objects, hands or other body parts, and even ones own face, in front of a camera. In this mode of interaction, the images are quite literally *performed* with the body.

Videos

We created and shared a number of videos online, to demonstrate the potential of such an approach. Frames from two of our more popular examples *Gloomy Sunday* (2017) and *We are all made of star dust #2* (2017) can be seen in Figures 5.14 and 5.15. For the first video, we use a normal webcam as live video input. For the second video however, we use a USB microscope. We discuss the significance of this in section 5.5: *Conclusion*.

Interactive installation

In addition to sharing the videos online, we also designed an interactive installation which has been exhibited around the world, allowing the public to interact with the system. Photographs of the installation can be seen in Figures 5.16 to 5.20

In the installation, we replicate the setting of the *Gloomy Sunday* video. We provide a large surface, covered in a number of every day objects such as broken cables and pieces of cloth. Users can play with these objects, and observe the output of the system — along with the overhead camera feed — on a large projection in front of them. The installation uses the models that we discussed previously: *Waves*, *Fire*, *Clouds*, *Flowers* and *Deep space*, and automatically cycles between them every 30 seconds.

The selection of the objects that we provide for the users to interact with, is very deliberate (Fig. 5.16). We found that providing one dark coloured cloth and one light coloured cloth, allows a user a wide range of opportunities with respect to what they can create. The cloths can be spread out to cover the entire surface, or they can be crumpled up into small lumps, or they can be twisted and bent into thin and long strips. Furthermore, providing both dark and light coloured cloths, grants a user the flexibility to control the dark and light regions of the image that they are creating. These are essential cues for the Neural Network, when it is constructing the output image. Just these two objects, can be combined in infinitely different ways. And most crucially, in ways that have the potential to be visibly radically unique, and thus can be interpreted by our system in radically different ways.

The cables provide additional opportunities to sculpt fine detail. Again we provide a mixture of dark cables and light cables. This allows the user to add details that either lighten or darken the underlying area. We also provide a mixture of thick cables — such as ethernet cables — and thin cables such as headphones. We cut off one end of each cable, as we found that sometimes having a large shape at the end of the cable is useful, while other times it is not. For example, in Fig. 5.17, we use the *white* headphone earpiece to stand in as the *lighter coloured* central area of a flower, while the underlying cloth becomes the petals. And in Fig. 5.18, we use the *black* headphone earpieces to stand in as the *darker coloured* central areas of the flowers.



Figure 5.14: Frames from the video *Gloomy Sunday* (2017)

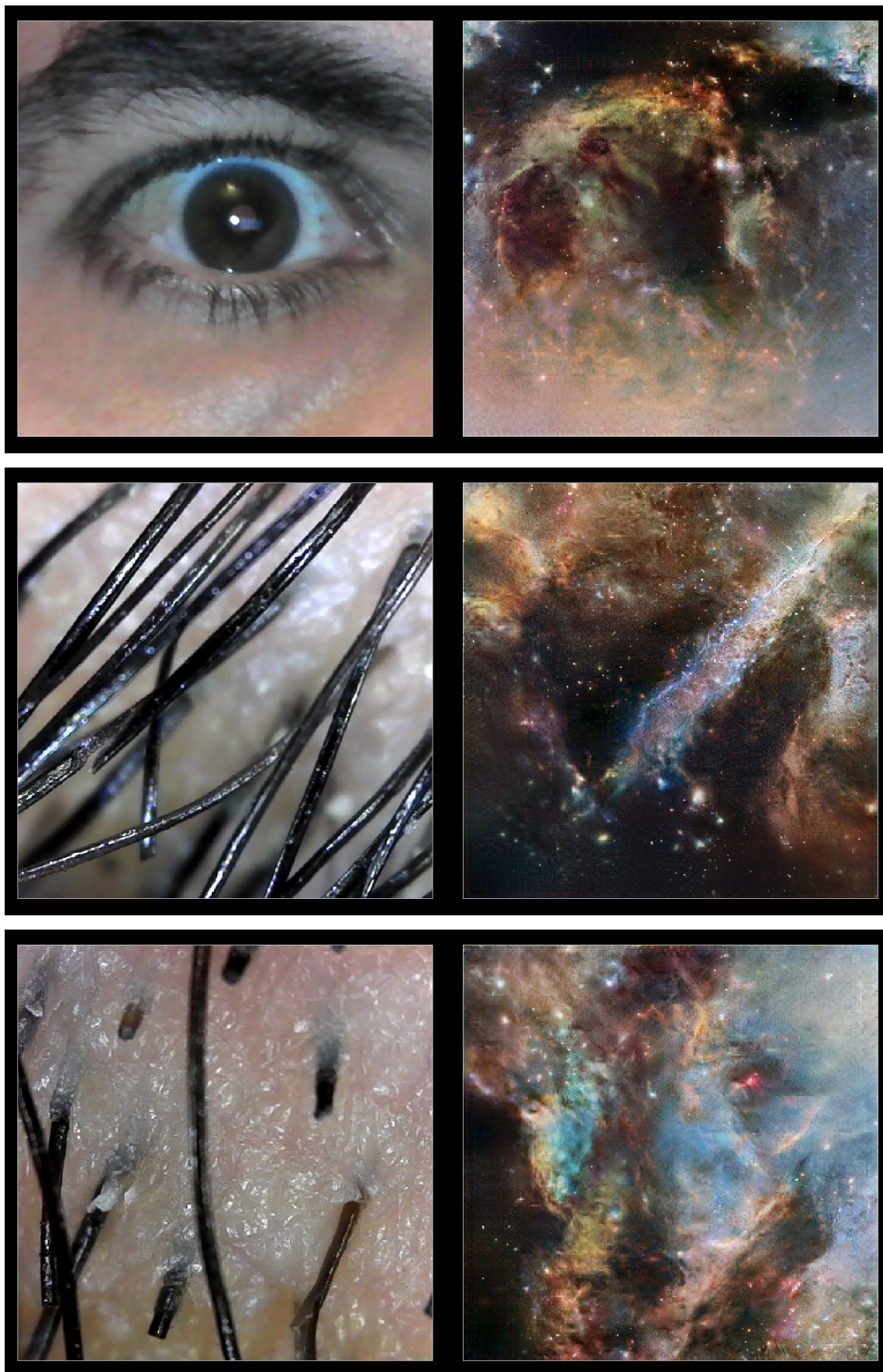


Figure 5.15: Frames from the video *We are made of star dust #2* (2017). The live video feed for this example comes from a USB microscope.



Figure 5.16: Objects provided with the *Learning to see* interactive installation. Users play with and manipulate these objects. The video feed from an overhead camera is fed to our system for processing.



Figure 5.17: *Learning to see* interactive installation at The Barbican's 2019 *AI: More than Human* exhibition in London, UK.

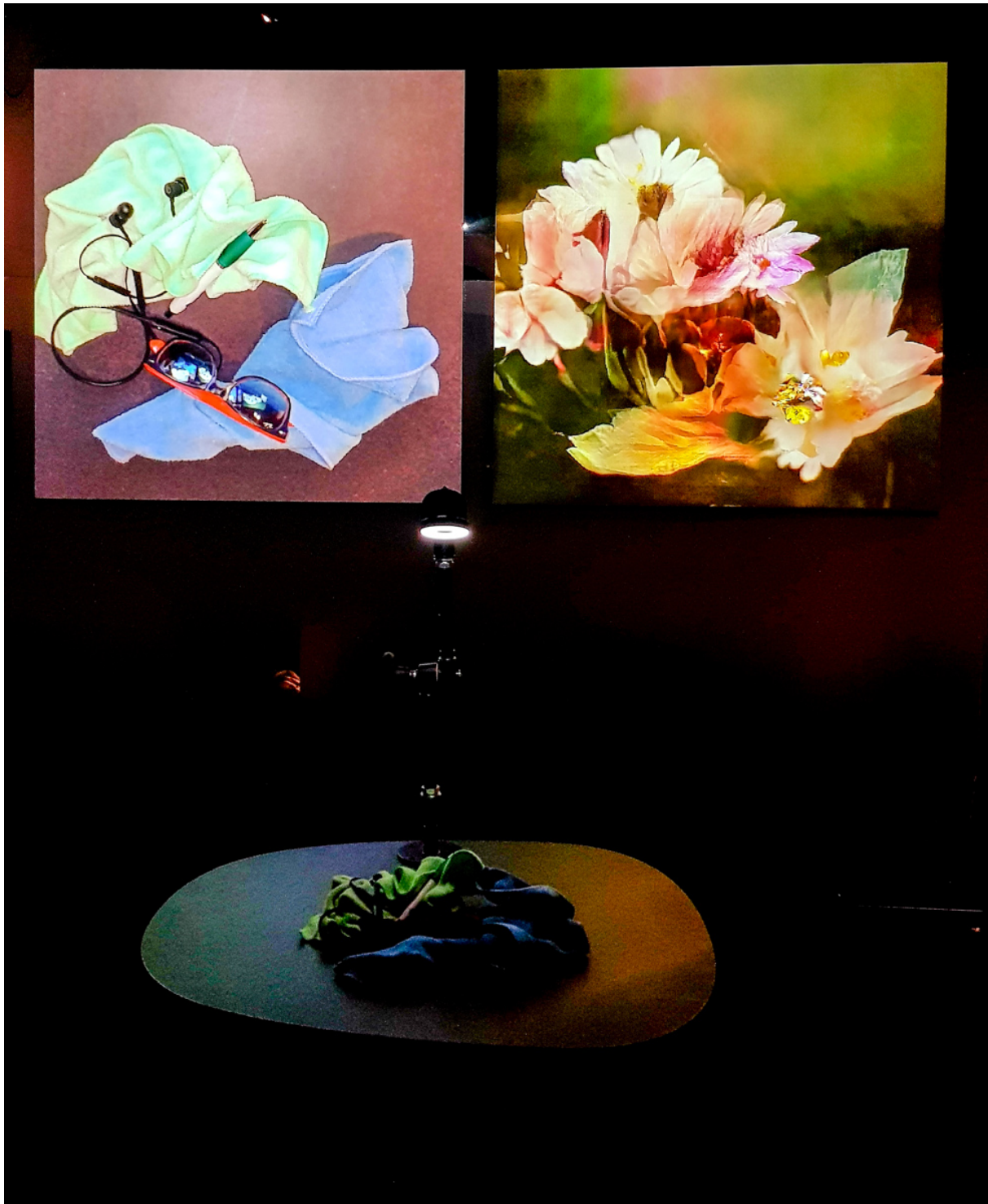


Figure 5.18: *Learning to see* interactive installation at The Barbican's 2019 *AI: More than Human* exhibition in London, UK.

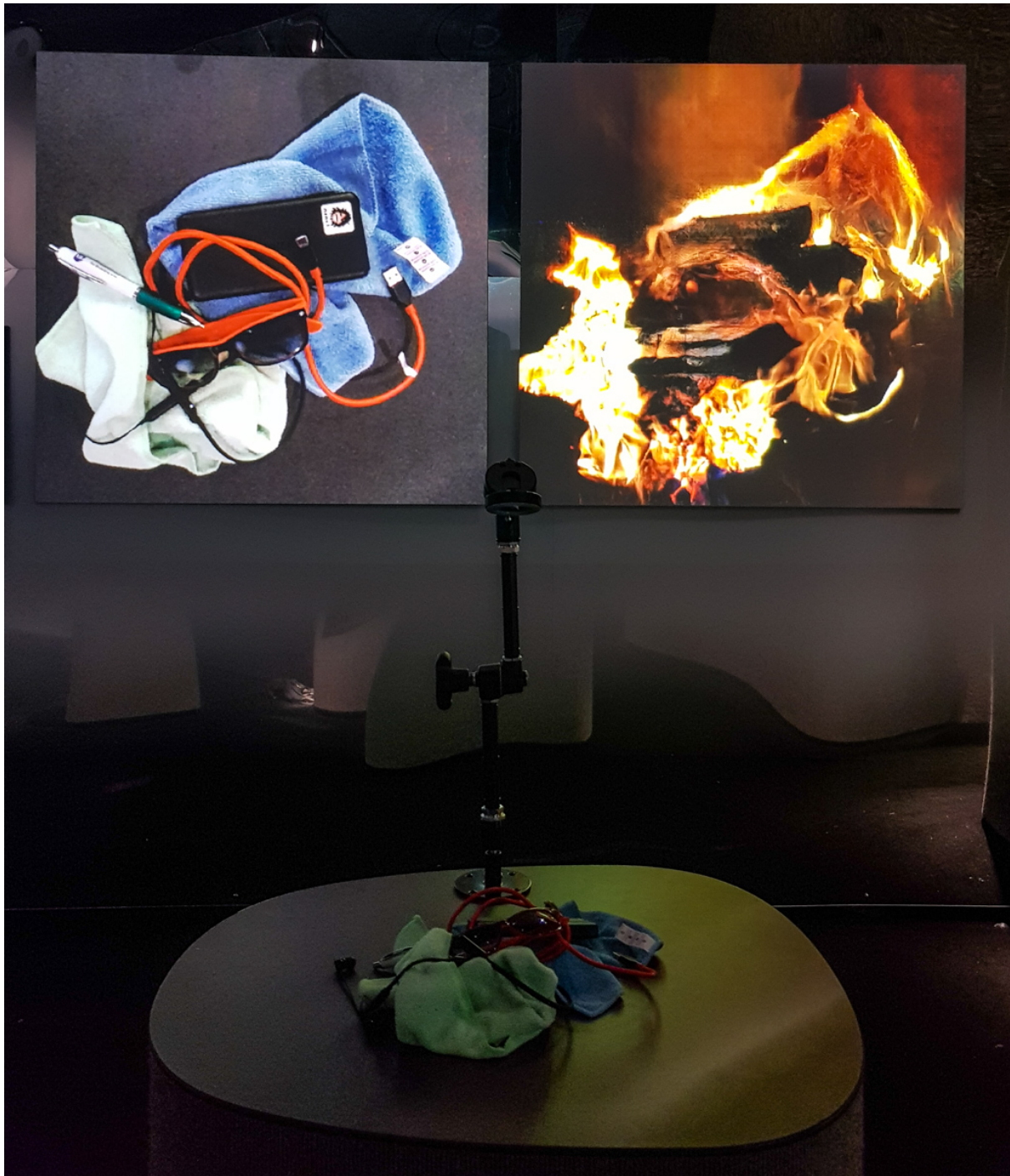


Figure 5.19: *Learning to see* interactive installation at The Barbican's 2019 *AI: More than Human* exhibition in London, UK.

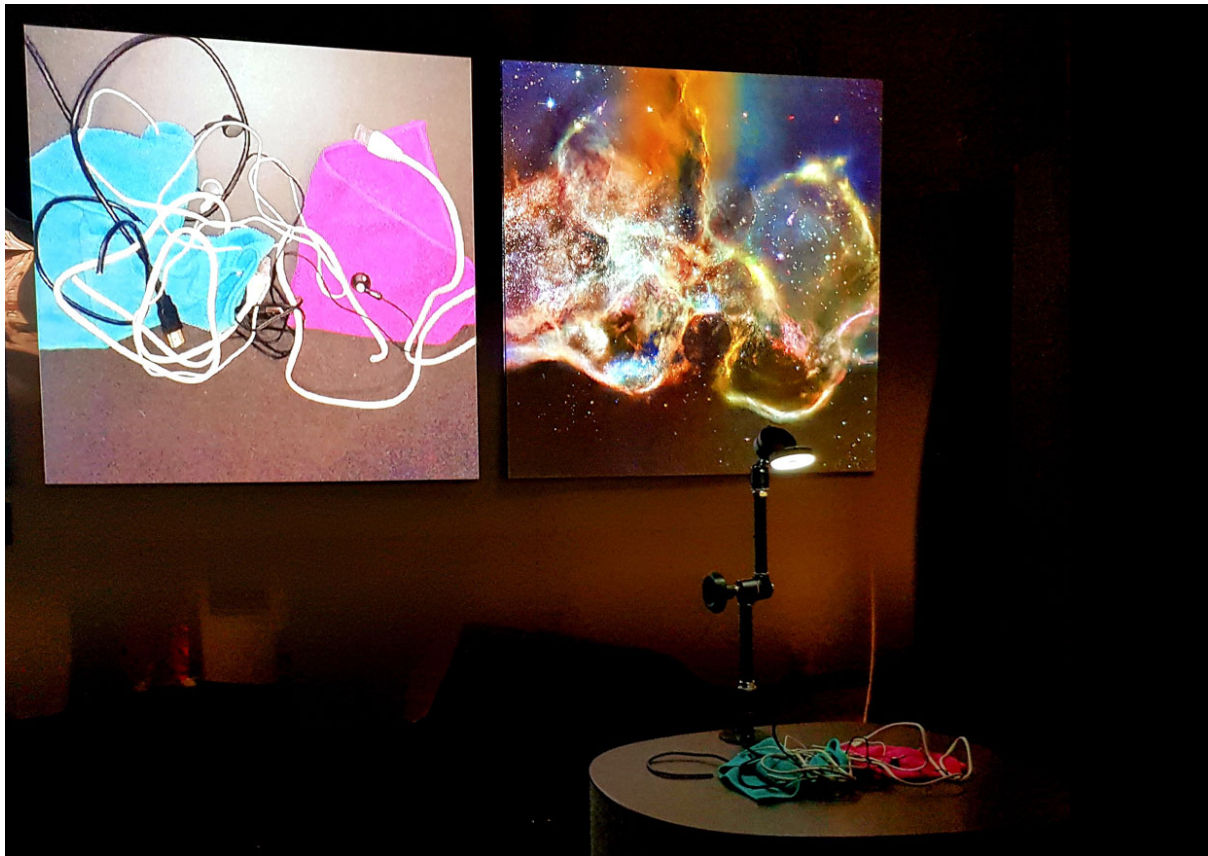


Figure 5.20: *Learning to see* interactive installation at The Barbican's 2019 *AI: More than Human* exhibition in London, UK.

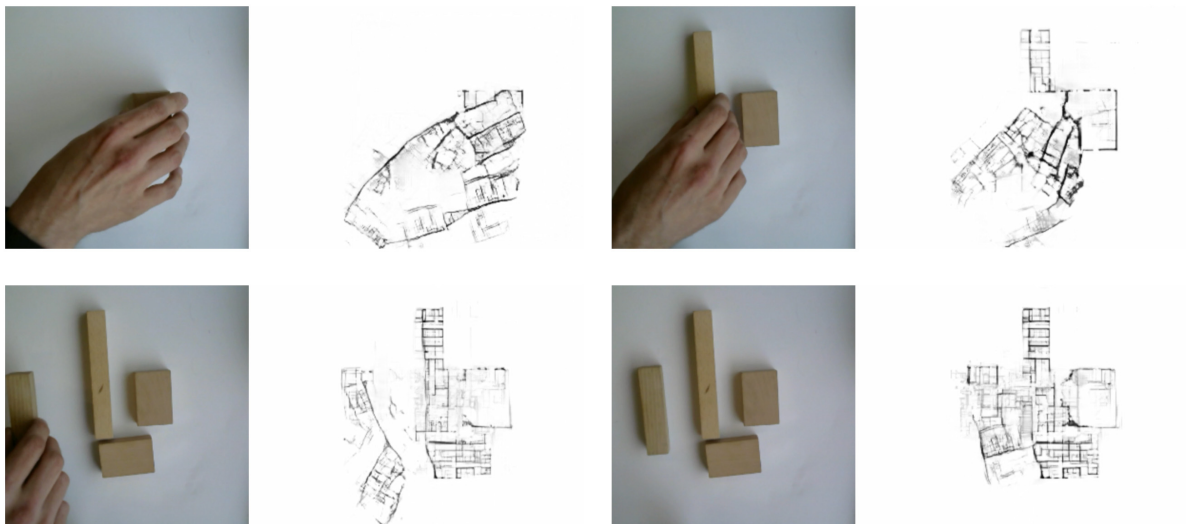


Figure 5.21: Frames from *Architectural machine translation* (video, 2019) by Swedish architect Erik Swahn. In 2019, inspired by the work that we shared in this area, Swahn started experimenting with a similar approach. He trained models on architectural plans and urban maps. Using wooden blocks, he was able to instantly prototype potential architectural diagrams.

5.4.3 Live parameter manipulation

In addition to *performing* some kind of movement in front of a camera — whether it be with hands, body, objects, drawing or any other means — one of the most powerful ways of interacting with our system, that allows us to really finesse the output, is by manipulating a set of *human-understandable* parameters in realtime.

In the previous two modes of interaction subsection 5.4.1: *Augmented drawing* and subsection 5.4.2: *Digital puppetry*, the control exerted by a user, comes from what is placed directly in the video feed. This provides the overall shape and structure of the image that will be generated by our software. Through *live parameter manipulation*, we allow a user more fine control, over how that particular shape and structure will be interpreted by the system.

In this context, by *human-understandable* parameters we are referring to parameters which a non-specialist person, unaware of the internal workings of a system, can quickly see and understand the effects of. This can be contrasted with parameters that influence a large number of variables, in a very non-linear, complex fashion. Often such parameters are very difficult, if not impossible, to describe in a simple language, or be understood by people who are not familiar with the inner workings of the system. For an interactive system to provide *meaningful* human control, any parameters that are available for a user to play with, must be *human-understandable*.

To give an example, imagine that due to some configuration of inputs, a generative system such as ours, produces an image of a nebula. It is very natural for a human user to want to *meaningfully modify* the image in certain ways. For example, they might want to make the nebula larger, or make it stretch out more, or make it more dense, or less dense, or make the shape of it more irregular, or smoother etc. If we apply transformations to the final image, they will affect all of the details as well. However, we wish to apply these transformations to the *structure* of the image, not the detail. Furthermore, the transformations that we wish to apply, are not necessarily simple geometric transformations such as scale, but might be the density of a nebula, or the intensity of a flame. The software that we have developed is not specific to nebula, or waves, or fire etc per se, but it provides exactly these types of controls, that can act not only on images, but on live video feeds.

With the method that we describe in the subsection 5.3.2: *Training* section, we train our models to effectively construct detailed output images, from abstract, blurry, blobby, greyscale input images. The image processing pipeline that we implement in our realtime inference software, processes the live video feed into such abstract, blurry, greyscale images. This is what allows our system to translate images of cables and cloth, into images of flowers, waves or nebula. We discuss how this works in more detail in the section 34: *Input abstraction* section.

The true power of our system however, comes from the fact that we can manipulate the parameters of this image processing pipeline in realtime. This allows us to control the way in which this abstraction takes place. The filters that we use in the pipeline are relatively simple filters, such as *blur*, *median*, *canny edge detection*, *adaptive threshold* etc. This means that they have a negligible overhead with regards to performance, and we can stack a large number of these filters, without comprising the framerate of the system.

Furthermore, while we have not conducted user surveys with regards to this, we hypothesize that users would be able to build an understanding of the effects of these parameters relatively quickly. This is especially reinforced by the fact that the system runs in realtime, and provides instant and continuous interaction. This allows users to play with the parameters, and without necessarily having any knowledge of the operations a median or canny filter actually performs, they can observe the results of their interactions instantly.

On the next four pages, Figures 5.23 to 5.26 show a selection of outputs from the *Waves*, *Fire*, *Clouds* and *Deep space* models respectively. Every one of these outputs was generated from the same input video frame, which can be seen in Fig. 5.22. In other words, the variation in the outputs in this section is not due to a variation in the *input video feed*. It is due to the *realtime manipulation of the image processing pipeline parameters* via the GUI.

We have frozen the video feed, so that the same image is processed and fed into the models in every example. This allows us to more accurately evaluate the effects of each parameter. Alternatively, if the video feed itself was also updating dynamically, it would be difficult for us to isolate the impact of changing the parameter values. Under normal circumstances, since the software runs at interactive rates (15-60fps depending on hardware), we can manipulate each of these parameters in realtime, while the video feed is updating. We can do this either via the GUI, or using an input device such as a midi controller, to allow a more *performative* interaction.

On these first Figures 5.23 to 5.26, we have collated a large selection of outputs from each model into a single image to aid side-by-side visual comparison. In the following pages, we show in more detail, how outputs such as these can be shaped via the parameters. In the captions below, parameter(s) which are modified from the previous screenshot, are indicated in **bold**.

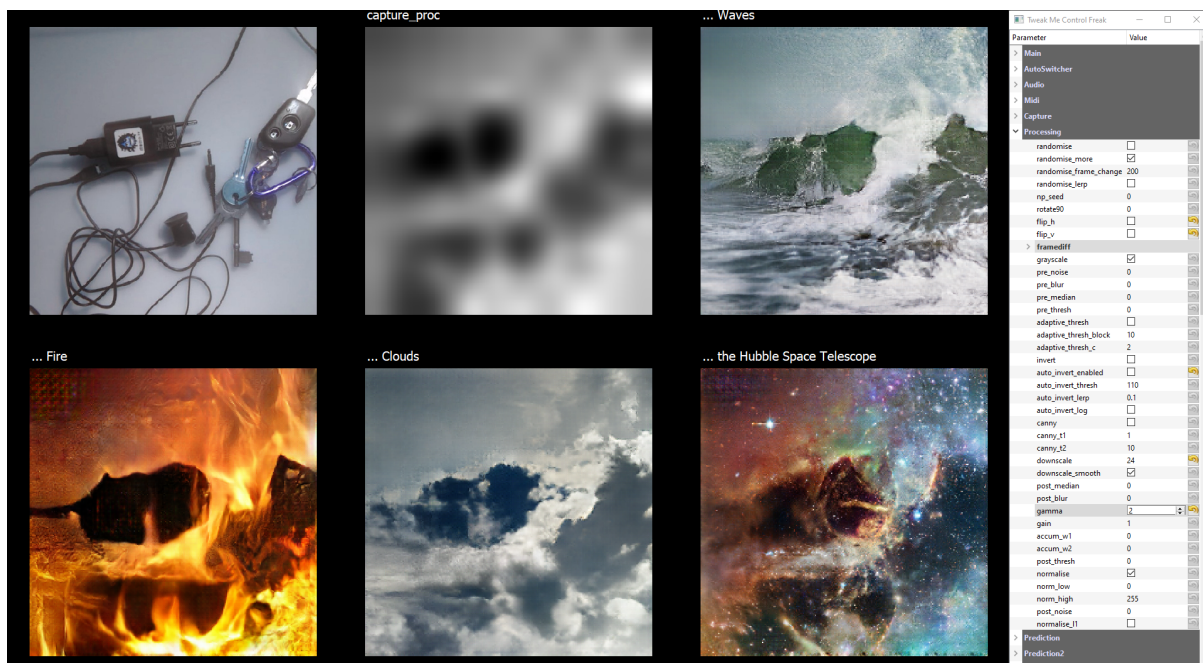


Figure 5.22: The image here in the top left shows the frozen video feed that will be used as an input for all of the subsequent examples in this section. All of the variations in output in the subsequent examples, come only from different configurations of parameters, which we will detail in each example.



Figure 5.23: A selection of outputs from the *Waves* model, collected in a single image to aid side-by-side visual comparison. Note that each of these images was generated from the *same* input video frame. Only the processing pipeline parameters are interactively modified in realtime via the GUI.

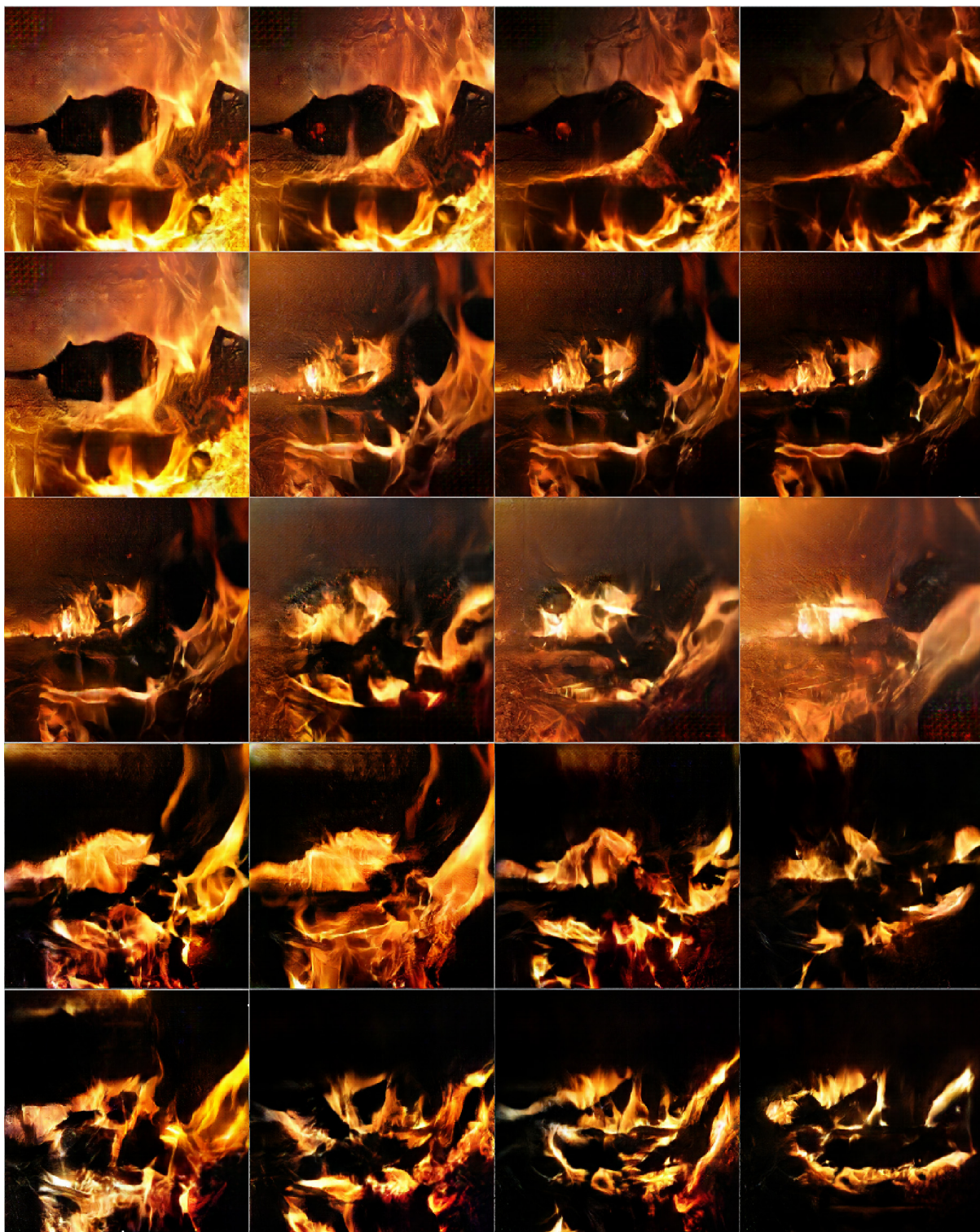


Figure 5.24: Same configuration as Fig. 5.23, with outputs from the *Fire* model.



Figure 5.25: Same configuration as Fig. 5.23, with outputs from the *Clouds* model.

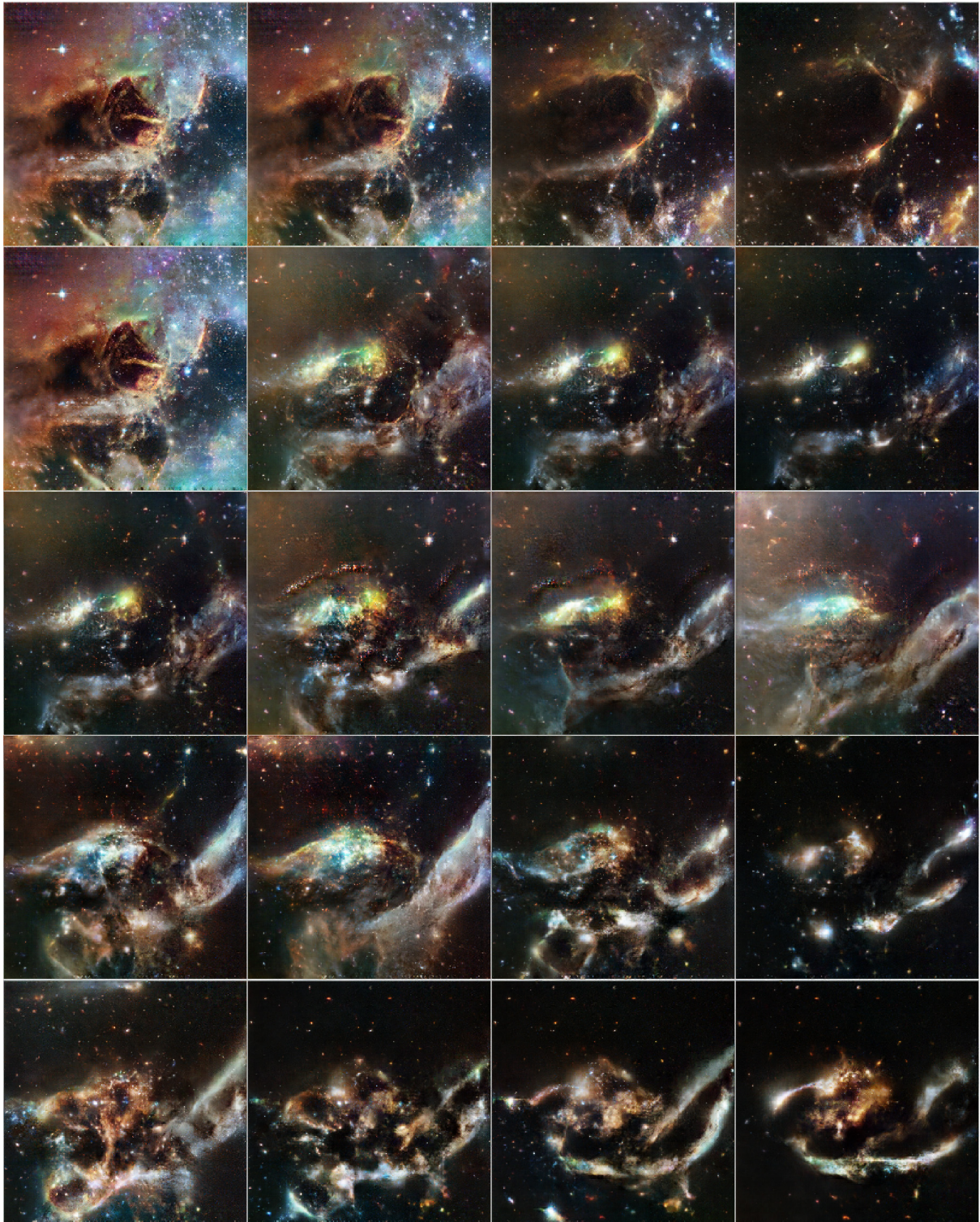


Figure 5.26: Same configuration as Fig. 5.23, with outputs from the *Deep space* model.

Gamma

As we increase *gamma* in Figures 5.27 to 5.29, the processed video feed *capture_proc* becomes darker. This results in the output images becoming darker as well. However, the darkening of the output, is visually very different from a simple post-processing effect. The *content* of the output is constructed by the Neural Networks in a manner that ensures the images are darker, while still maintaining the qualities of the training data. For example, the waves have less foam, fire becomes less intense, clouds become thinner, the nebula becomes sparser.

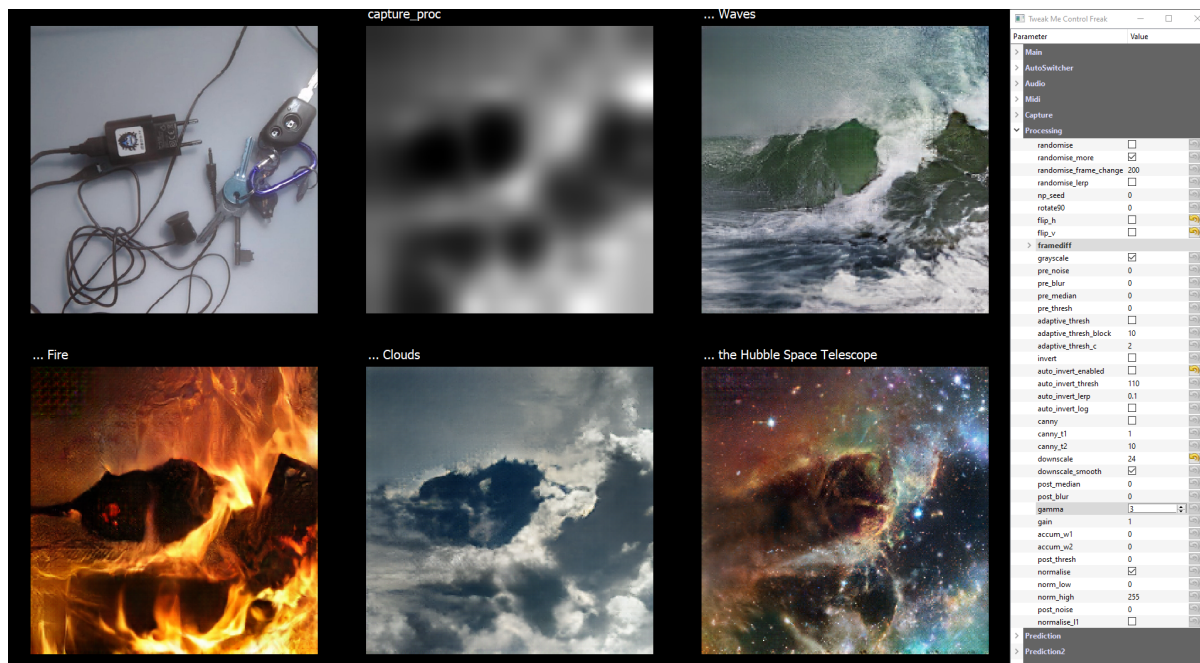


Figure 5.27: *downscale=24, gamma=2.*

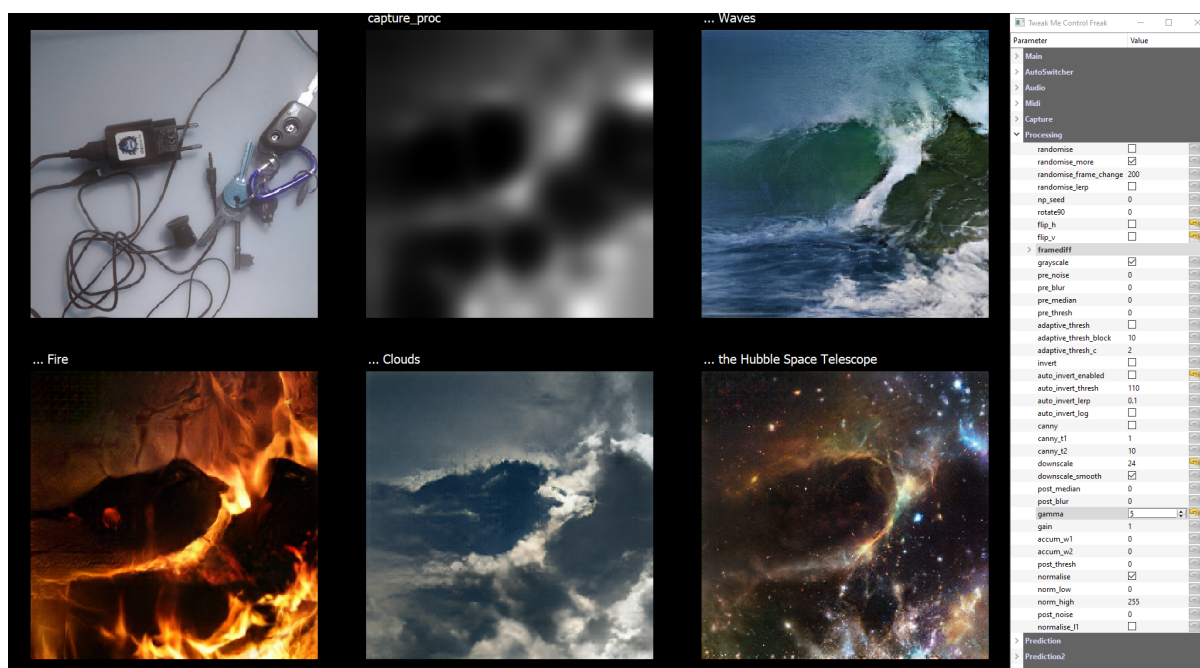


Figure 5.28: *downscale=24, gamma=5.*

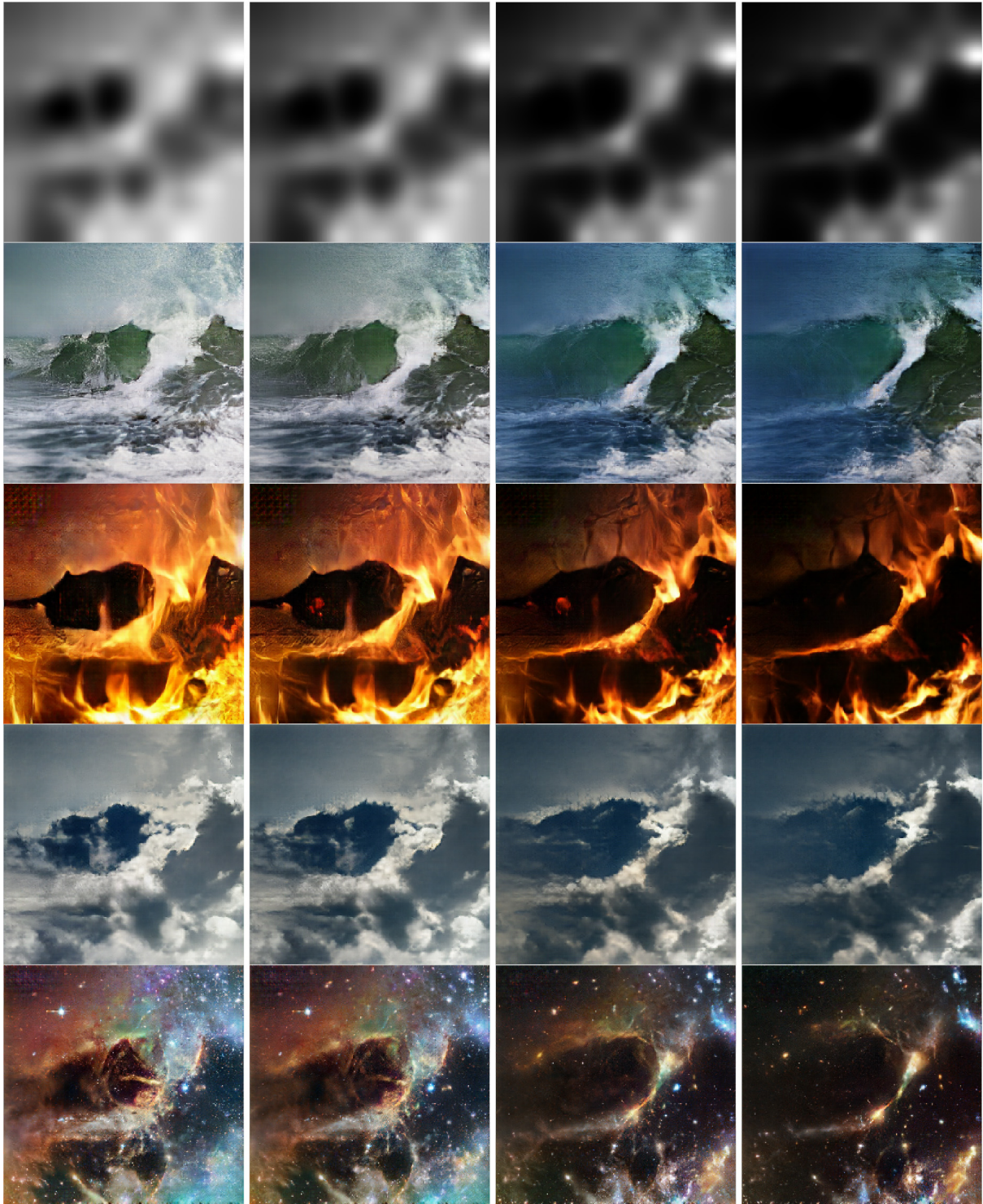


Figure 5.29: More examples of modifying γ , arranged in a single image to aid side-by-side visual comparison. Each of the columns show the outputs for $\gamma=2, 3, 5$ and 7 respectively, with $\text{down-scale}=24$ for all.

Invert

When we enable *invert*, the output images are constructed such that the light and dark areas are swapped. However, the *details* of the output images are not inverted, only the *structure* is. For example, what was previously *foam* becomes *water*, *flames* become *ember*, *clouds* become *empty sky*, *nebulars* become *empty space*, and vice versa (Fig. 5.30).

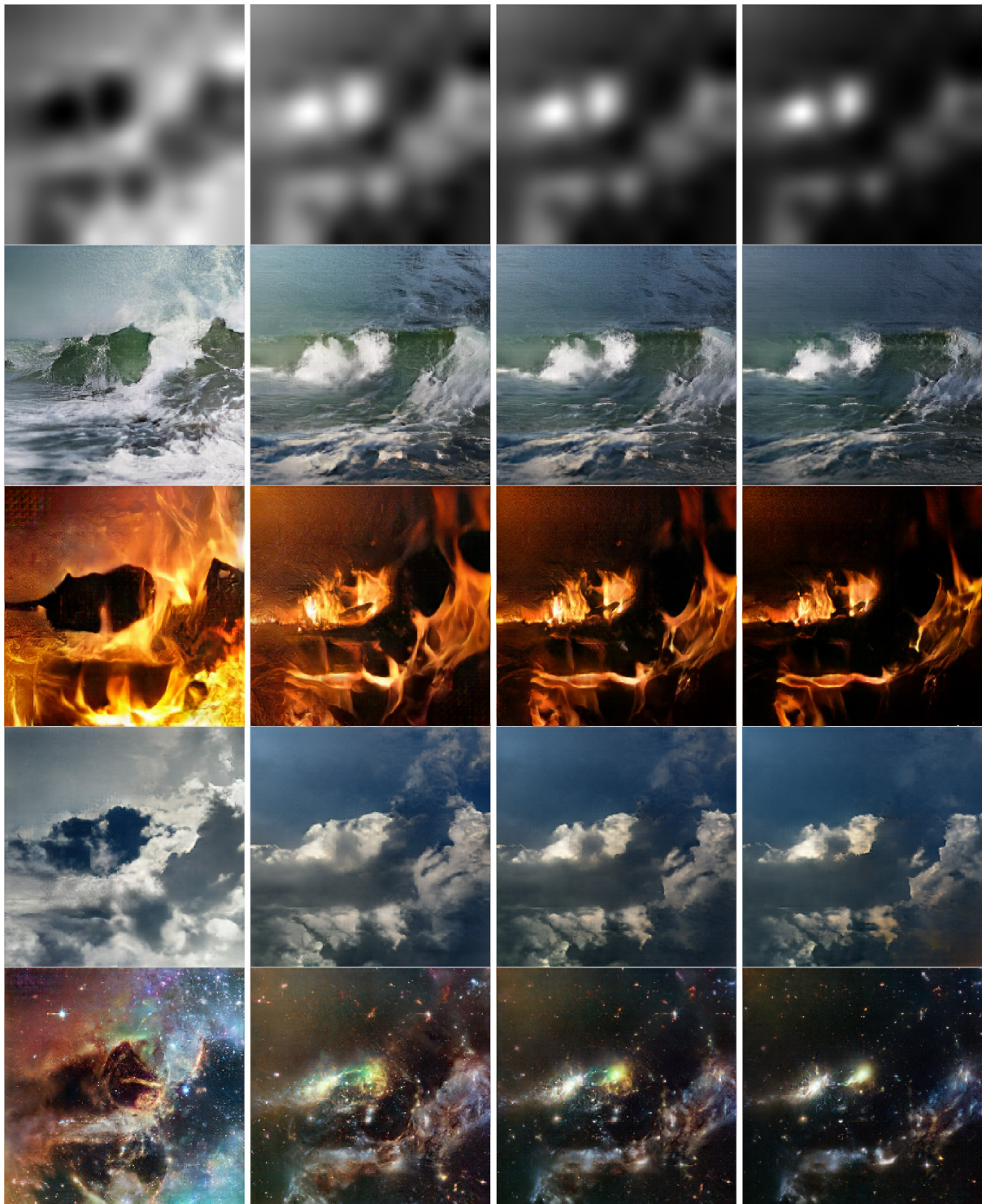


Figure 5.30: The first column is for reference, and shows the outputs with *invert* turned off and *gamma*=2. The following three columns show the outputs with *invert* turned on and *gamma*=1, 2 and 3 respectively.

Median

We can use a *median* filter to smooth the outlines of the shapes in the processed video feed. This results in outputs that have smoother structures. Adjusting the amount of *median* controls the shapes of the structures in the output images. Increasing the amount, results in smoother structures. Since these filters apply to the input image only, they do not compromise the level of detail and quality of the output images. Examples can be seen in Figures 5.31 to 5.33.

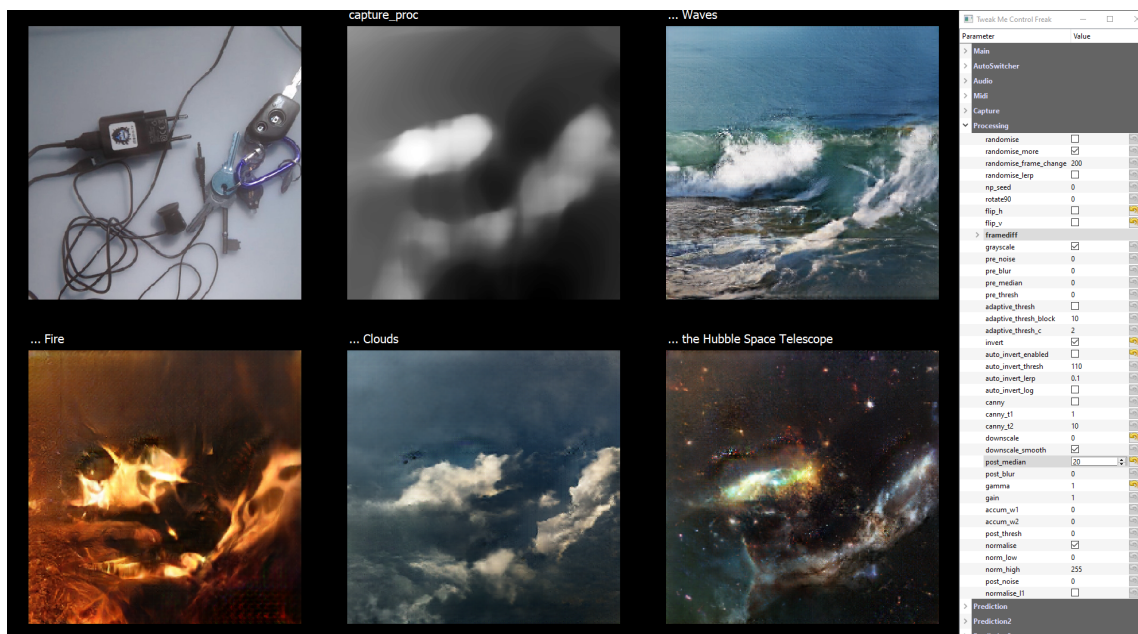


Figure 5.31: $invert=True$, $downscale=0.$, $post_median=20$, $gamma=1$.

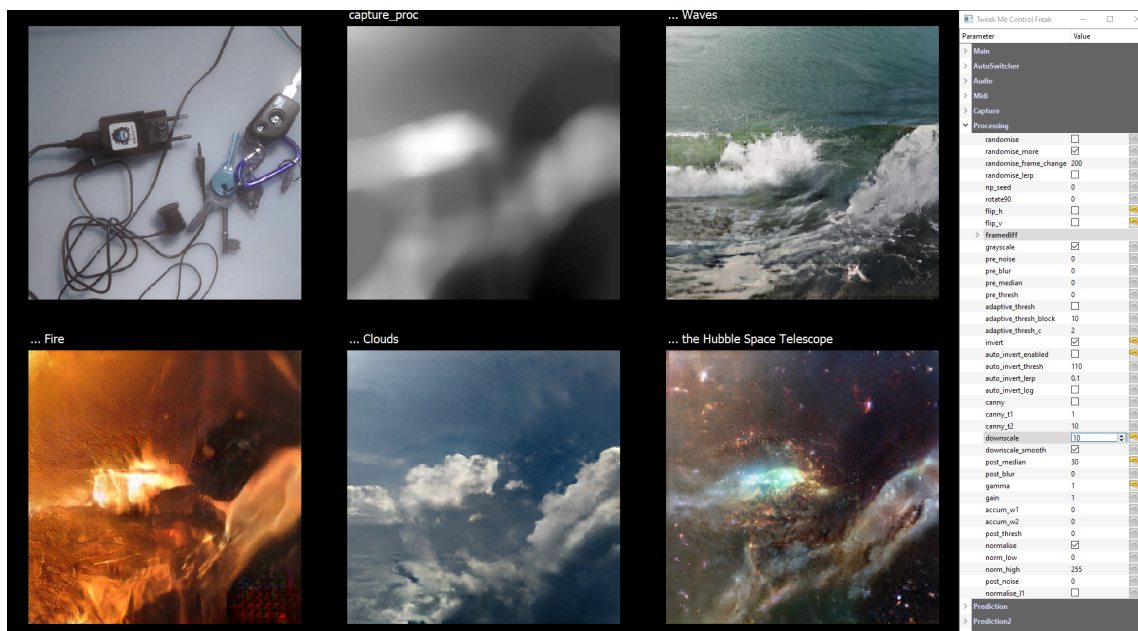


Figure 5.32: $invert=True$, $downscale=10$, $post_median=30$, $gamma=1$. In the previous example, to demonstrate the effect of the *median* filter, we disabled the *downscale* low-pass filter. This results in undesirable high frequency artefacts in the output images. To address this, we reintroduce a small amount of *downscale*.

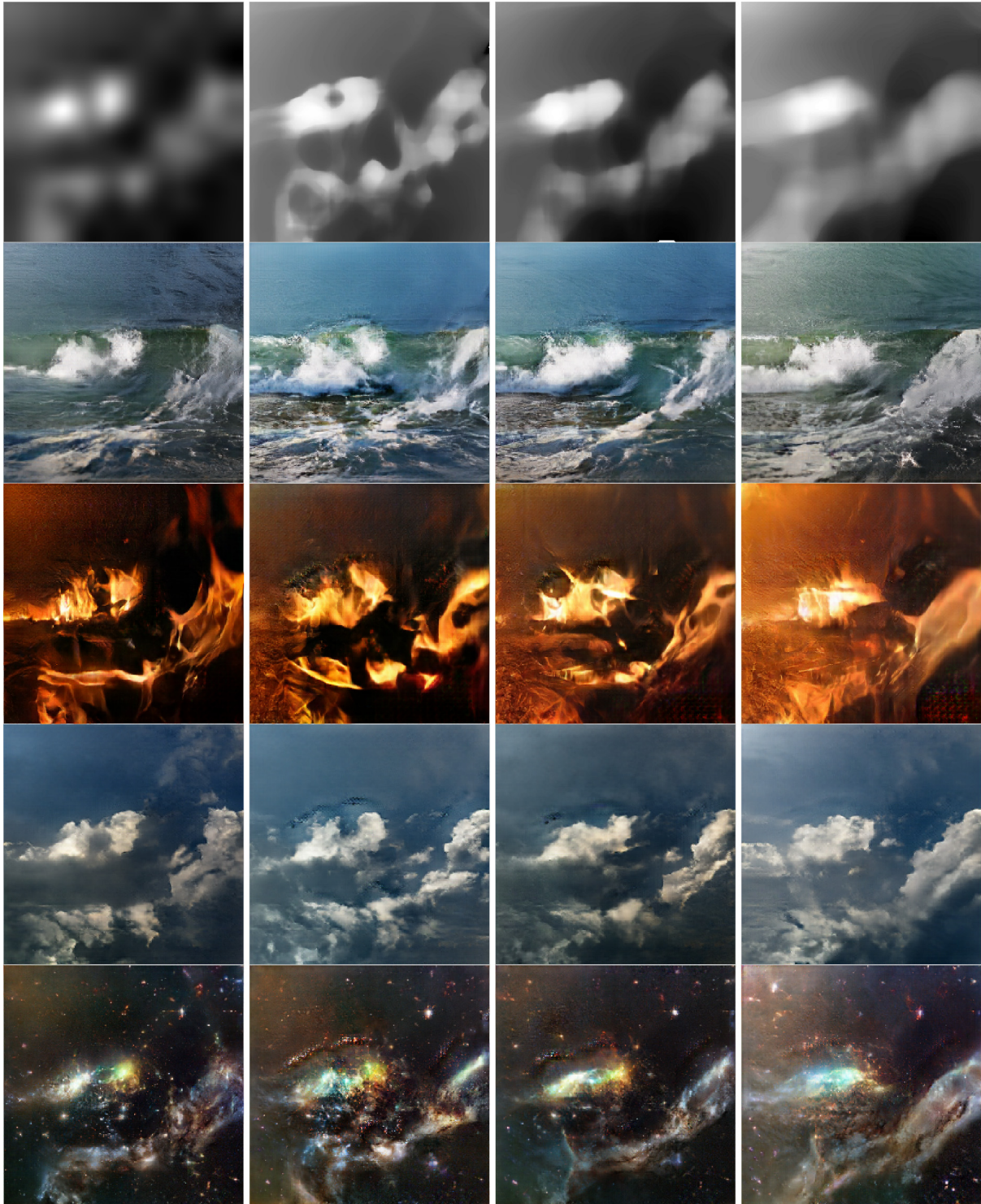


Figure 5.33: More examples of modifying the amount of the *median* filter, arranged in a single image to aid side-by-side visual comparison. The first column is for reference, and shows the outputs with *median* turned off, $\gamma=2$ and $\text{downscale}=24$. The following three columns show the outputs for $\text{median}=10$, 20 and 30 respectively. The last column also has $\text{downscale}=10$ to remove the high frequency artefacts, while the second and third columns have *downscale* turned off.

Adaptive threshold

In the examples in Figures 5.34 and 5.35, we use an *adaptive threshold* to convert the input to a binary image, emphasizing the edges.

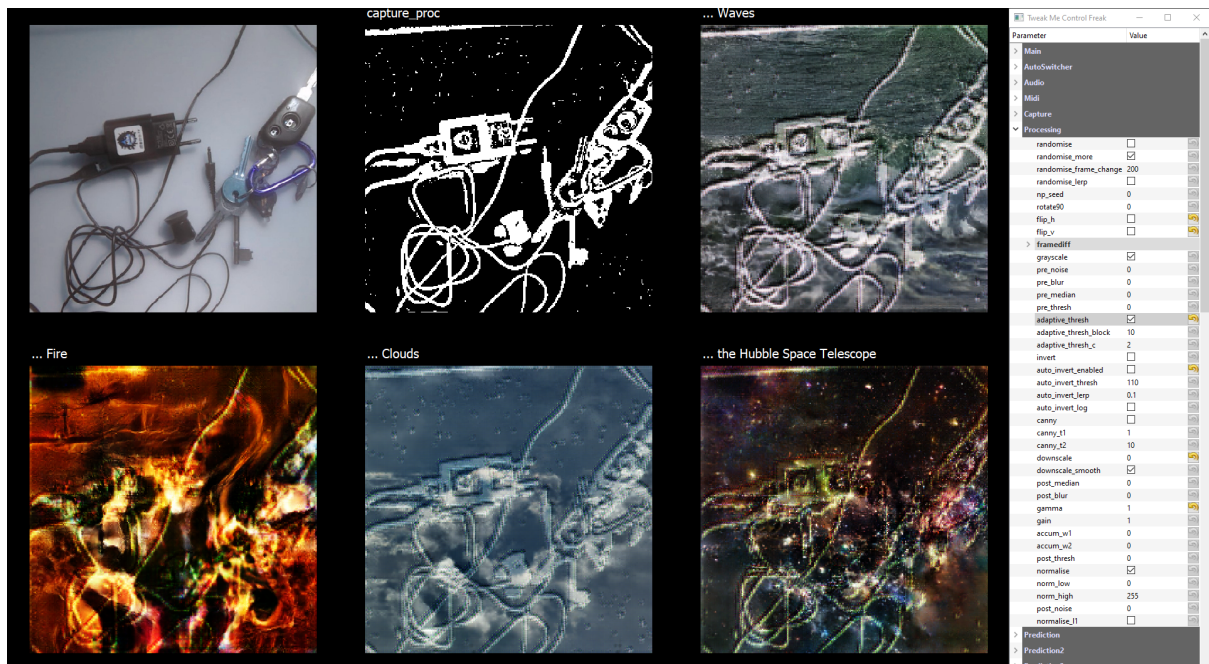


Figure 5.34: *adaptive_thresh*=True, *downscale*=0. For each pixel, we set a threshold equal to a gaussian-weighted sum of the pixel's neighbours within a block, minus a constant C. If the pixel value is greater than this threshold, it is set to black, otherwise it is set to white. This behaviour can be inverted with the *invert* parameter. The block size and constant C can be adjusted with the *adaptive_thresh_block* and *adaptive_thresh_c* parameters respectively.

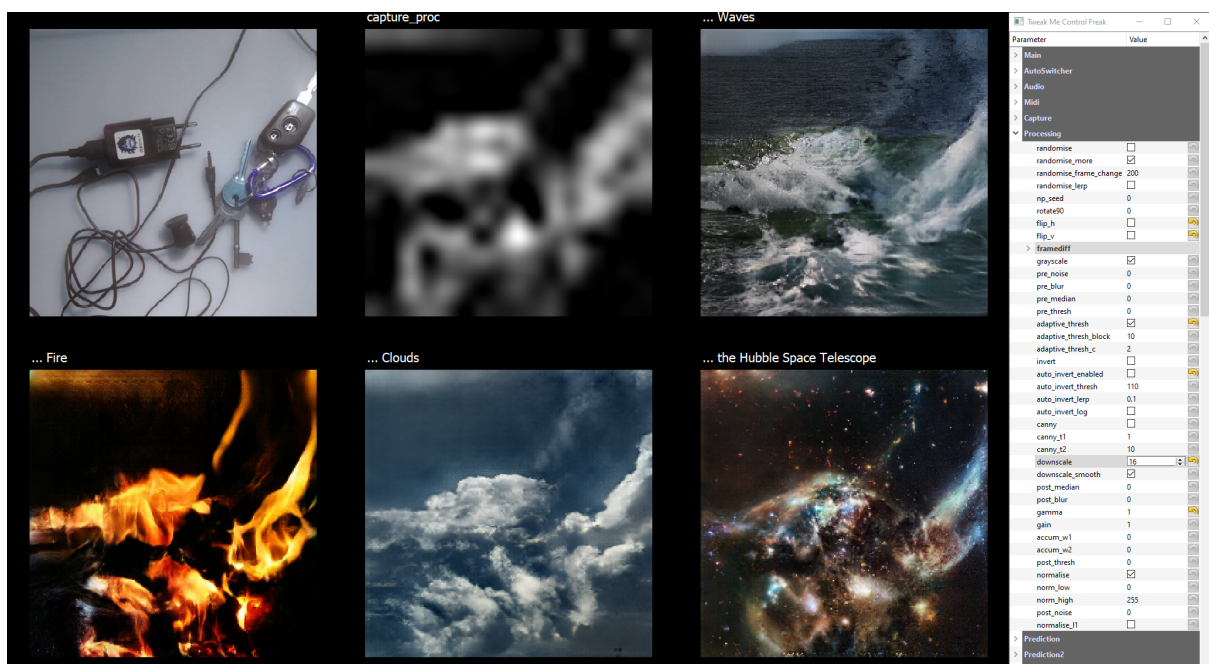


Figure 5.35: *adaptive_thresh*=True, *downscale*=16. In the previous example, we disabled the low-pass filter, to demonstrate the effect of the adaptive threshold. The undesirable high frequency artefacts which arise as a result of this can be resolved with a low-pass filter such as *downscale*.

Post vs pre-median

In the examples in Figures 5.36 and 5.37, we apply a *median* filter *after* the *adaptive threshold*. Since the median is applied at the end of the image processing pipeline, we call it *post-median*.

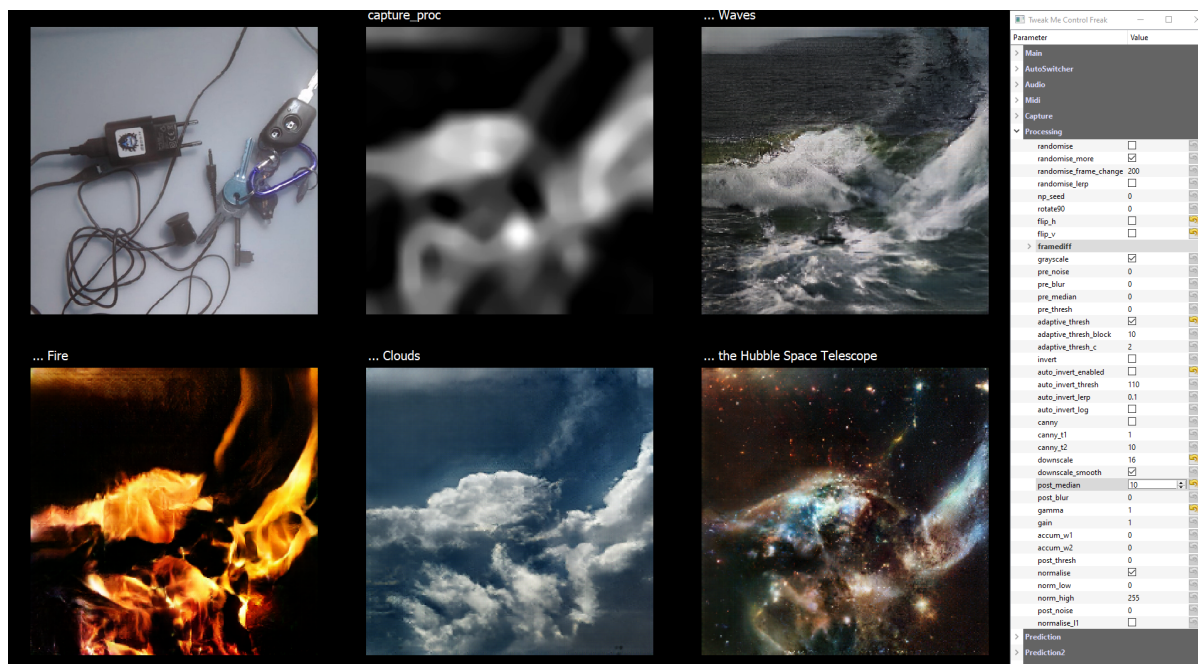


Figure 5.36: $adaptive_thresh=True$, $downscale=16$, $post_median=10$. We can reintroduce a *median* at the end of the image processing pipeline, smoothing the edges of the shapes.

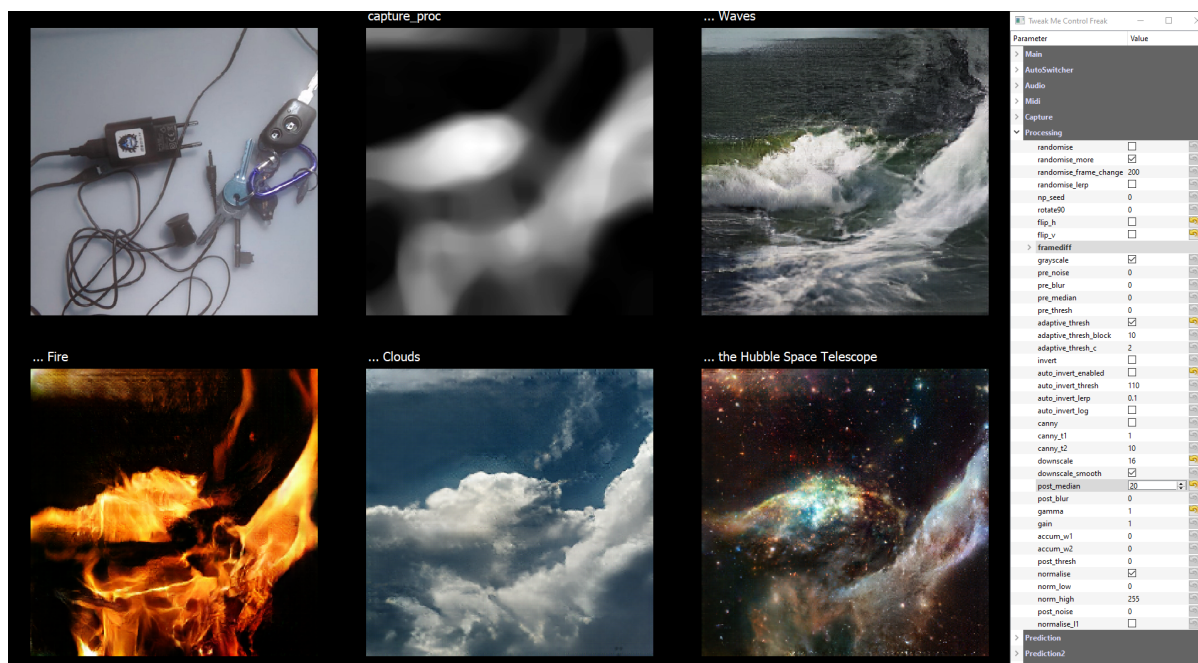


Figure 5.37: $adaptive_thresh=True$, $downscale=16$, $post_median=20$. Increasing the amount of *median* smooths the shapes of the structures even more.

The median filter can also be applied at the start of the image processing pipeline. We call this *pre_median*. Unsurprisingly, this has a significant effect on the images generated, as can be seen in Fig. 5.38.

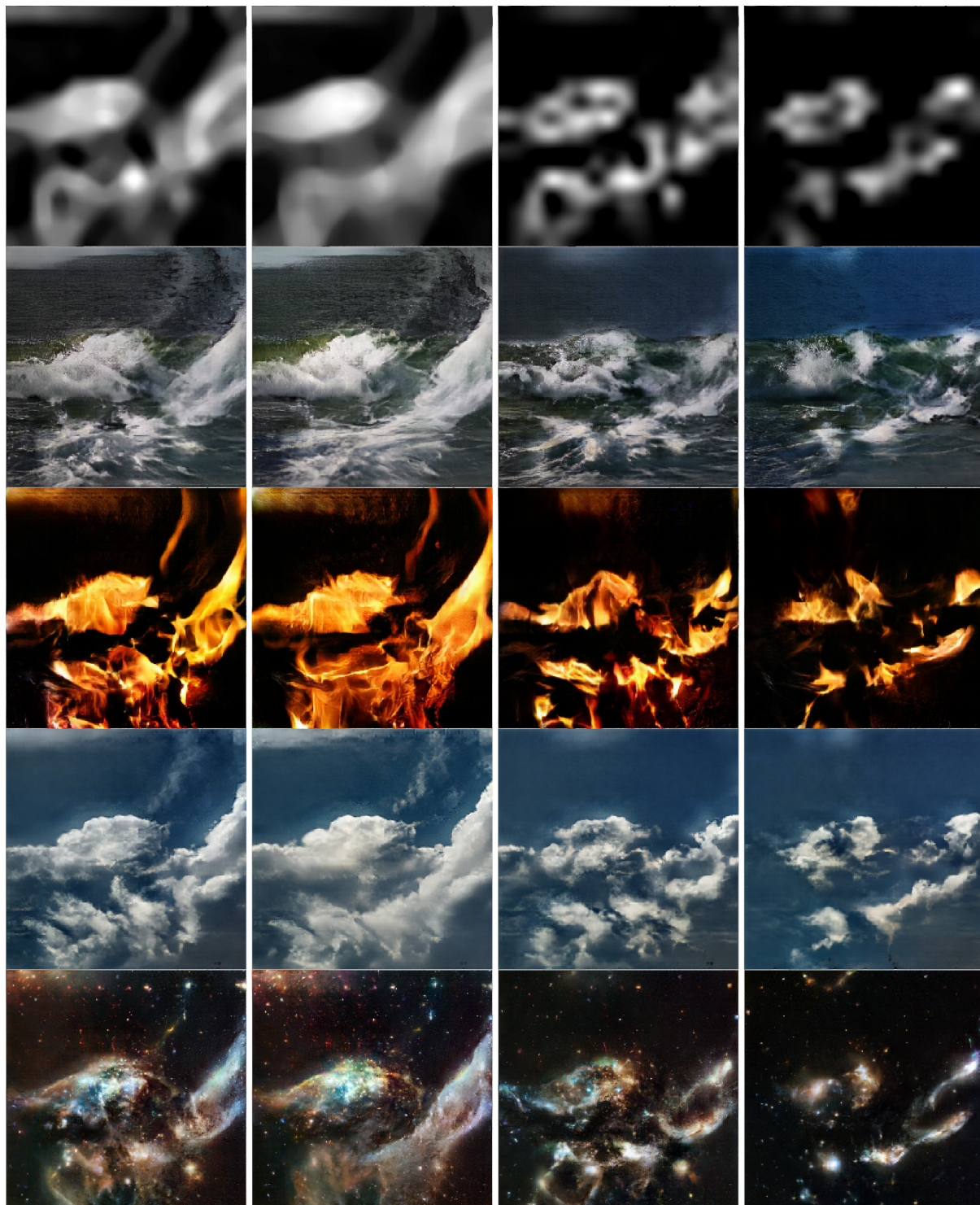


Figure 5.38: The first two columns show the outputs for *post_median*=10 and 20 respectively. The last two columns show the outputs for *pre_median*=10 and 20 respectively. In all cases, the *adaptive threshold* and *downscale* settings are the same as the previous examples.

Canny edge detection

We can also use a *canny edge detection* algorithm. The end result of this is similar to the *adaptive threshold* that we used in the previous examples, in that it produces a binary image emphasizing the edges. However, the resulting image has much greater detail and the lines are generally thinner. In Figures 5.39 to 5.43, we show the effects of playing with the canny thresholds, and simultaneously enabling other filters.

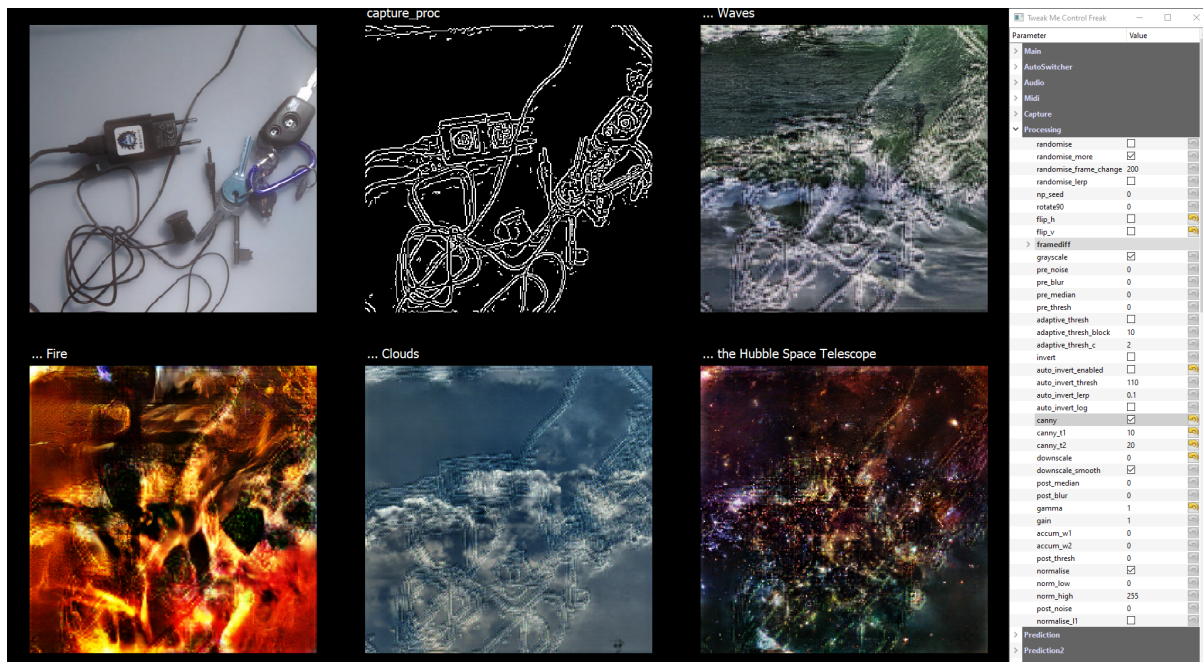


Figure 5.39: *canny*=True, *canny_t1*=10, *canny_t2*=20.

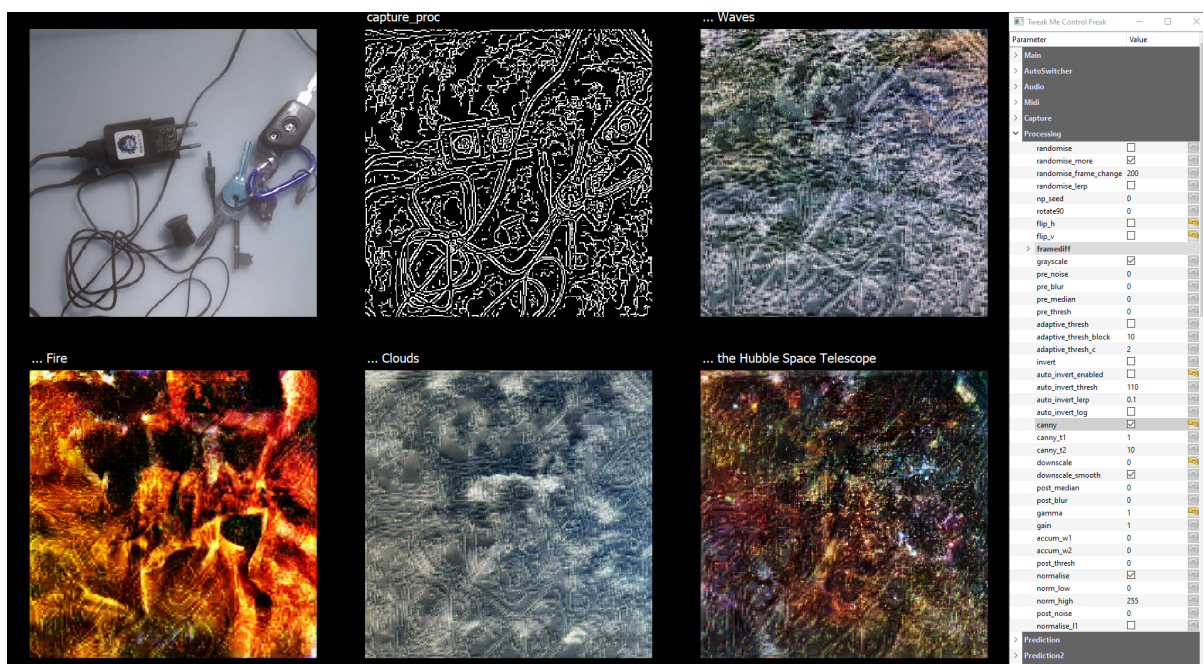


Figure 5.40: *canny*=True, *canny_t1*=1, *canny_t2*=10. We can adjust the threshold values for the canny algorithm using *canny_t1* and *canny_t2* to increase or decrease the amount of detail captured.

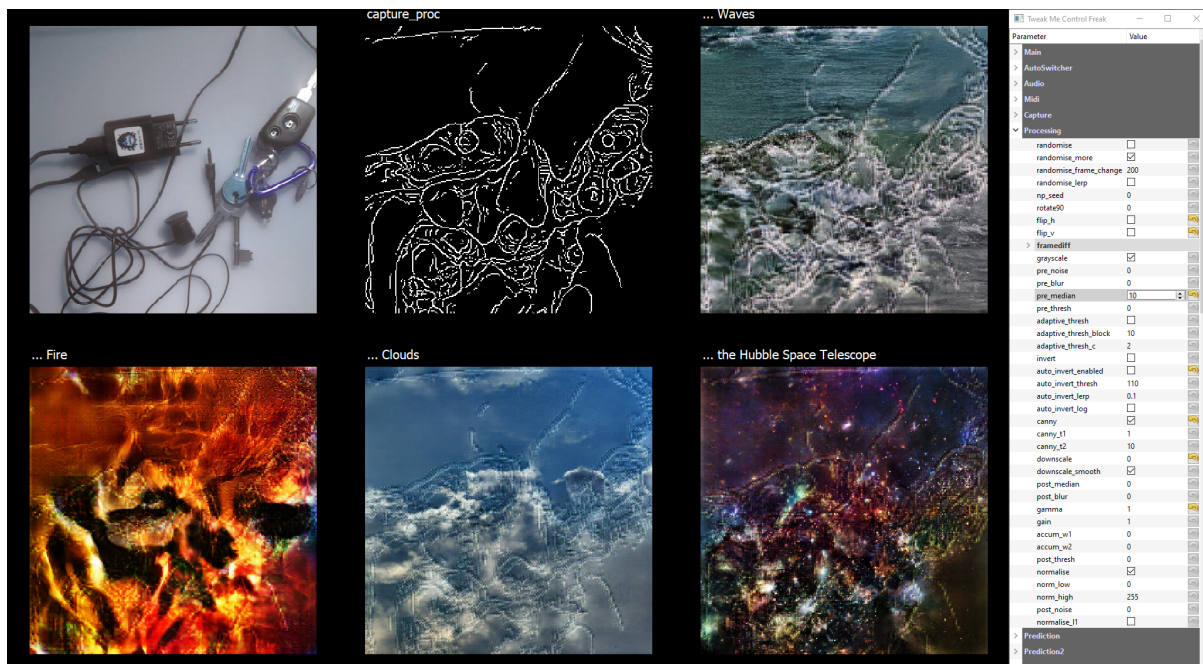


Figure 5.41: $pre_median=10$, $canny=True$, $canny_t1=1$, $canny_t2=10$. Applying a *median* filter before the canny filter, removes some of the noise and smooths the edges detected by the algorithm.

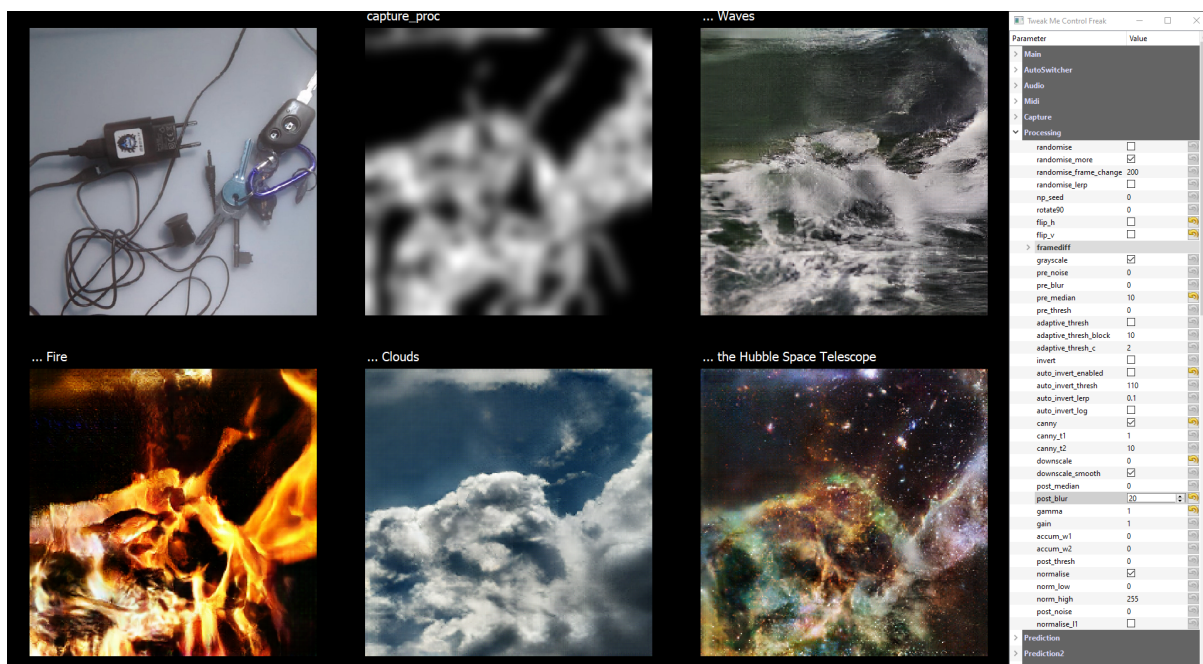


Figure 5.42: $pre_median=10$, $canny=True$, $canny_t1=1$, $canny_t2=10$, $post_blur=20$. As we demonstrated in the previous examples, to remove the high frequency artefacts in the output images, we can apply a low-pass filter to the input image. In the previous examples, we used *downscale* as a low-pass filter. Another option that we have available, is to use a *blur*. The difference between *post_blur* and *downscale* is very subtle, but for the sake of variety, we will use *blur* in these next few examples. Similar to *median*, we can apply the blur at the start of the filter pipeline, or at the end. In this particular case, since we would like to filter the results of the canny edge detection, we use *post_blur*.

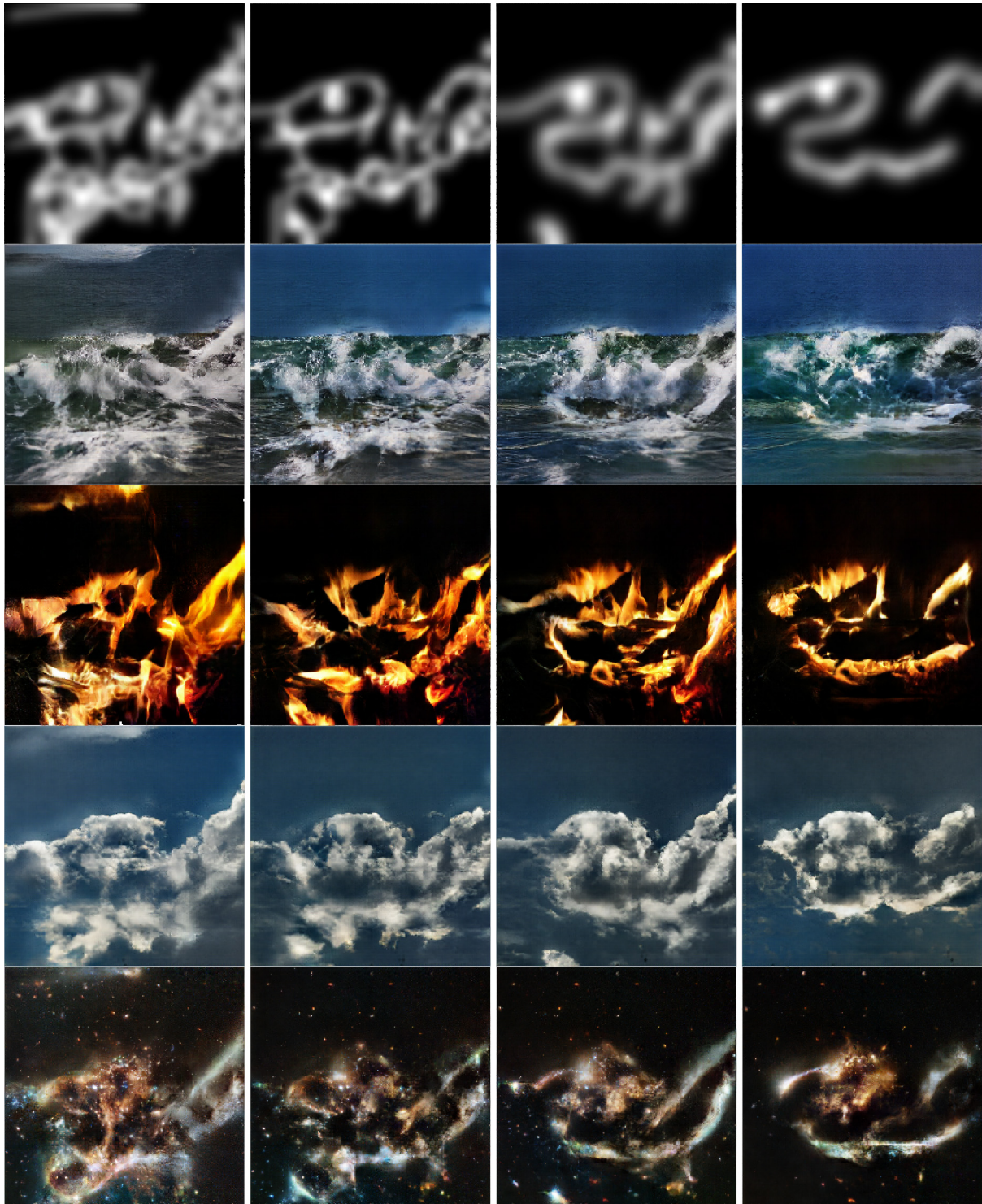


Figure 5.43: More examples of modifying the *canny* thresholds and the amounts of *pre_median* and *post_blur* filters. The $(pre_median, canny_t1, canny_t2, post_blur)$ settings for each column are as follows. 1:(10, 5, 20, 20); 2:(10, 10, 50, 20); 3:(10, 10, 100, 30); 4:(15, 10, 100, 30).

5.5 Conclusion

In this chapter, we presented a *live video feed processing* system which leverages the capabilities of generative Deep Neural Networks trained on large datasets of images, while allowing users to expressively interact with the system, and meaningfully control the output in realtime.

Our system allows interaction via the video feed, either through subsection 5.4.1: *Augmented drawing* or subsection 5.4.2: *Digital puppetry*. Interaction is also possible via subsection 5.4.3: *Live parameter manipulation*, which is an additional, very powerful aspect of our system. Through these modes of interaction, users can play with our system, to shape, conduct, steer, and create compositions, images and videos in a realtime, expressive manner. In this respect, the method that we demonstrate in this chapter, addresses every concern that we set out to investigate in this thesis.

It is worth noting that a user of our software does not have explicit control over how objects, or sections of objects, that are presented to the camera, are interpreted by the system. In other words, it is not possible to explicitly instruct the software, that a particular *input object A*, should be interpreted by the system as a particular *output object A'*, and another particular *input object B*, should be interpreted by the system as a particular *output object B'*. For example, we cannot instruct the software to treat large patches of cloth as flower petals, and headphone earpieces as the central area of the flower, as we mentioned in the section 5.4.2: *Interactive installation* section. Even as the developer of the software, *we* do not have this level of control. This is a decision that the Neural Network makes on its own, based on everything that it has been trained on, the entirety of the video feed that is being presented to it, and the configuration of the image processing pipeline. In fact, in the case of the headphones and flowers in Fig. 5.17 and Fig. 5.18, we did not know that this would happen. We *discovered* it, through *playing* with our software. We discuss this in more detail in the final chapter section 7.3: *Summary of contributions and outcomes*.

Furthermore, to provide this level of granular control is not our goal. To be able to provide that level of granular instructions to the software is not necessarily very difficult. But it would require domain-specific configurations, such as semantic color maps or similar. In other words, for every dataset and model, we would need to provide some kind of metadata, mapping characteristics of input objects and image features, to output objects and images features.

Instead, our aim is to devise a versatile prototype that can *automatically* work with and adapt to *any* dataset, with no additional configuration necessary. The most important aspect that we take into consideration, is the *learning curve of the user*. A user can start playing with our software, with zero knowledge to begin with. As the models automatically switch between *Waves, Fire, Clouds, Flowers, Deep space*, and potentially any other model, users do not need to relearn anything new. All of the knowledge and experience that they gained from working with one dataset and model, can be transferred and adapted to other datasets and models.

An additional feature that allows our system to express such a wide range of outputs with relative ease, is that the system provides *realtime, continuous* interaction and control. This allows us to experiment, observe and learn how the system responds to our actions. This in turn allows us to respond to the system accordingly, and shape the results as we desire. In

the case of the headphone-flower example that we mention above, we instantly noticed how the system was creating flowers when the headphone earpiece was positioned in the centre of a large area of cloth. We were then able to exploit this knowledge to our advantage.

Observing the general public interacting with our interactive installation, we found that some people might spend just a few seconds playing with the objects, before smiling and walking away. However, many people also spent very long periods of time, many long minutes, as they meticulously arrange their perfect bouquet of flowers; or shape their perfect wave, or nebula.

It is also worth adding, that instead of cables and cloth, we could use objects especially designed for this purpose. For example, we could use objects that physically resemble flowers, even before they are processed by the Neural Network. Or we could use specially manufactured objects that are more malleable and provide greater opportunities to be bent and shaped by the user. Undoubtedly, this has the potential to provide more realistic outcomes. However, one of our aims with this work, is to demonstrate how the most mundane and ordinary house-hold objects, such as broken cables and used pieces of cloth, can become perfect tools of creativity, given the appropriate algorithmic treatment.

We produced many videos with our system, and we shared them online. These videos were widely circulated on both mainstream and social media. They were included in academic articles and books, and even shown during Nvidia CEO Jensen Huang’s keynote at GTC (GPU Technology Conference) 2019¹⁴. These videos captured the imagination of a diverse range of people from many different fields. These include AI researchers, artists, designers, architects, cognitive scientists, neuroscientists, physicists, journalists as well as members of the general public. Having presented this work at many cross-disciplinary conferences, and through informal conversations with many individuals from these fields, it is clear that there is a general interest in this kind of work that demonstrates the potential of *human-AI collaborative co-creation*. In other words, in *AI-augmented human creativity*. And *expressive, continuous, realtime, meaningful* interaction is at the heart of this human-AI creative relationship.

We would like to add, that we believe our primary contribution is not necessarily the *technical specifics* of what we presented in this chapter. In the realtime inference software that we developed, we hard-coded a particular image processing pipeline that we feel covers a wide range of use cases, and has allowed us to shape the output in a very diverse and meaningful way. We presented many examples of these in the section 5.4: *Experiments and results* section. We did spend significant amounts of time, trying to design an image processing pipeline and set of ‘magic values’, that produced subjectively both the *nicest looking*, and *most diverse* results. We believe we have produced what we consider to be a very powerful and flexible system.

However, we present these details, as merely a starting point, which can undoubtedly be dramatically improved upon, especially when tailored to a user’s specific needs or vision. For example, while we were developing and testing our software, we would often encounter situations where we envisaged a particular output image. Or to be more precise, we envisaged particular *modifications* to an image generated by our system at a particular moment in time. However, these modifications were not possible with the image processing pipeline that we had implemented at that point in time. We would often realize, that the desired modifications would

¹⁴Nvidia GTC 2019 Keynote: <https://www.youtube.com/watch?v=Z2XlNfCtxwI&t=32>

be possible, by for example moving the median filter *before* the canny filter instead of *after* it. Since we are authors of the software, we could quickly make the required code modifications, and achieve the desired results within a few seconds. However, undoubtedly, a more powerful software tool, could have such capabilities to add, remove and even write new filters at runtime, built into the GUI. In that respect, we do not claim that the software tool that we have created, is ready to be deployed and used by artists and designers around the world, allowing them to create and shape any image that they can imagine. For our work in this thesis, as opposed to extensively developing one single software tool we prefer to investigate many different methods and develop each method enough to demonstrate its potential to the extent that it can ideally provide a foundation for future research. We believe that this may potentially have more impact and encourage more research in this particular field of continuous, Meaningful Human Control of Deep Neural Networks for creative expression.

One additional point worth discussing, is whether it's really necessary to have such a *manually designed, hand-crafted* image processing pipeline in front of the Deep Neural Network. After all, a Deep Neural Network is effectively a very deep image processing pipeline. And one of the major motivations behind Deep Learning, as we discussed in section 1.2: *Why Deep Learning?*, was to avoid hand-crafted feature engineering. So could we omit our hand-crafted image processing pipeline, and instead *learn* a desired pipeline end-to-end? In other words, could we train a Deep Neural Network in such a way, that it *learns* these meaningful, human-understandable parameters — and potentially many more? The short answer is yes, most probably, but there are some complications involved which we discuss with regards to potential future directions in chapter 7: *Conclusion*.

We would like to end this chapter with some reflections on higher level motivations behind this work, and in fact, also behind the work that we will present in the following chapter.

In addition to the primary questions that we seek to address in this thesis with regards to Realtime Continuous Meaningful Human Control, we are also interested in exploring computational models of intelligence and cognition, as an *analog* for how humans make meaning.

As we mentioned previously, we focus our examples on five models, trained on five specific datasets that we collected during our research. These are *Waves, Fire, Clouds, Flowers* and *Deep space*.

The selection of these particular datasets and models are very deliberate, and tie into this higher level motivation. These categories, represent the five classical elements of nature: water (*waves*), fire, air or sky (*clouds*), earth (*flowers*) and the aether or the void (*deep space*). While this list varies slightly from culture to culture, many ancient civilizations have constructed a list of very similar basic elements, which they use to try and make sense of the world. These basic elements are seen as the foundational building blocks of all of reality.

In the examples that we present in this chapter, Artificial Neural Networks look out onto the world, onto objects such as broken cables or pieces of cloth, or scribbles on a piece of paper. They try to ‘make sense’ of what they are seeing. But they can only make sense of what they are seeing, in context of what they have seen before.

They can see only through the filter of what they already know.

Just like us.

Because we too, see things not as they are, but as we are.

A Deep Neural Network is effectively a highly complex *filter* that can receive any input — even white noise — and transform it into an output that resembles the data that it was trained on. The Artificial Neural Network, in effect, projects its life experience, onto the inputs that it is receiving.

And this is how we make meaning. We yearn structure. We look for patterns, and we find them, and we project what we know onto them. We look at stars in the night sky, and recognize familiar shapes. So we invent stories, and we believe them. We invent rituals. We see the future in coffee cups, in tea leaves, in crystal balls, in dreams. It's how we deal with uncertainty, and how we connect with each other. We see things not as they are, but as we are. Our perception of the world is internal to us, and depends on who we are, where we've been and what we've done. And this is not limited to ancient superstitious practices. It happens every moment of every day of our lives.

In this context, the term *seeing*, refers to both the low level perceptual and phenomenological experience of vision, as well as the higher level cognitive act of *making meaning*, and constructing what we consider to be truth. Our self affirming cognitive biases and prejudices, define what we see, and how we interact with each other as a result, fuelling our inability to see the world from each others' point of view, driving social and political polarization. The interesting question is not only “when you and I look at the same image, do we see the same colors and shapes?”, but also “when you and I read the same article, do we see the same story and perspectives?”.

Everything that we see, or read, or hear, we try to make sense of by relating to our own past experiences, filtered by our prior beliefs and knowledge. In fact, even these sentences that I'm typing right now, I have no idea, what any of it means to you. It's impossible for me to see the world through your eyes, think what you think, and feel what you feel, without having read everything that you've ever read, seen everything that you've ever seen, and lived everything that you've ever lived.

Empathy and compassion are much harder than we might realize, and that makes them all the more valuable and essential.

Furthermore, when we consider the *Deep space* model, and that this Neural Network has been trained on images from the Hubble Space Telescope, we might be tempted to think that this model has learnt what deep space looks like. However, that is not entirely accurate. The Hubble Space Telescope, does not provide raw snapshot photographs, as *objective visual representations* of deep space. Rather, it streams numbers. And then those numbers are interpreted, by us, humans, by engineers, by scientists, even by artists. We stitch together patches, composite layers, and even add colours. We don't necessarily know exactly what distant space actually looks like, or whether it is even possible for our minds to fathom those kinds of scales, hundreds of thousands of light years in diameter. This Neural Network is not learning what deep space really looks like. It is learning what *we think* deep space *should* look like, tainted with our assumptions, our biases, our romantic visions.

Speaking of romantic visions, there is one additional motivation specifically behind the

“We are made of star dust” series of works, and in using a microscope (Fig. 5.15) with this particular model. It is fascinating to remind ourselves, that the atoms in our bodies, and in fact in everything that we see around us, were forged in the dying hearts of stars and supernova, billions of years ago, billions of light years away, and spewed across the galaxies, before they came together, for a very brief moment in time, to become you and me. To quote the astrophysicist Neil deGrasse Tyson: “We are all connected. To each other, biologically. To the earth, chemically. To the rest of the universe atomically.”. And Carl Sagan: “The cosmos is within us. We are made of star-stuff. We are a way for the universe to know itself”.

Chapter 6

Deep Meditations: Latent storytelling

6.1 Introduction

The over-arching theme of this thesis, is to investigate methods that grants a user Meaningful Human Control over generative systems that leverage the capabilities of Deep Neural Networks. In the previous chapters, we have presented a number of studies that explored this in a Realtime Continuous manner, as this is an important point of focus for our research. In this chapter however, we investigate a *non-realtime, non-continuous* method.

We recognise that non-realtime, non-continuous modes of interaction have traditionally played an important role in existing, established creative workflows, particularly in narrative time-based media, such as the production of long-form video or film. Using this as a starting point, we develop a system based around a typical *Non-Linear Video Editing* (NLVE) workflow, and we present a method for *long-form story telling* within a deep generative model’s space of latent representations.

In other words, we develop a system that gives users the ability to *discover* and *design* desirable *trajectories* in the high-dimensional latent space of a generative model, allowing them to construct *stories* to produce *long-form videos*, with meaningful control over the narrative.

We demonstrate our system by creating a number of films. These include a 60 minute multi-screen video installation called *Deep Meditations: A brief history of almost everything in 60 minutes*, which we have shown at venues such as St James Hatcham Church Gallery (London, UK), Foster & Associates’ nine storey atrium in ME by Melia Hotel in London (UK), the Zen Buddhist Temple Kennin-ji in Kyoto (JP), the Mori Art Museum (Tokyo, JP), and Sonar+D Festival in Barcelona (ES) (Fig. 6.1).

We have also used our system in a collaboration with the renowned electronic musician *Max Cooper* for his seminal audio-visual performance and tour *Yearning For The Infinite* which premiered at *The Barbican* in London (Cooper & Akten, 2019). The piece we made for the performance, *Morphosis*, was also released online as a music video in 2020 (Akten & Cooper, 2020). We presented a paper on this research at the *Machine Learning for Creativity and Design Workshop* at *Neural Information Processing Systems* (NeurIPS) in 2018 (Akten et al., 2018).



Figure 6.1: Photos from *Deep Meditations: A brief history of almost everything in 60 minutes* at Sonar+D Barcelona, 2019

6.2 Background and motivations

In subsection 2.1.3: *Latent manipulations*, we discuss a number of methods such as semantic vector manipulations, and latent interpolations, that allow some level of control over the output produced from a generative model. These techniques allow control over a *single sample* produced by a generative model. In the case of an image based model such as a DCGAN, this translates to allowing control over a *single image* produced by a generative model. Effectively, these methods allow us to find interesting and desirable *points* in latent space.

At the time that we developed the work that we present in this chapter, in 2017–2018, we are not aware of any research or tools which granted users *any* kind of control over *long-form videos* produced via image-based deep generative models such as DCGANs. All DCGAN based *video work* at the time, were either *random trajectories* sampled from the latent space of a model, colloquially referred to as *random walks in latent space*; or very short semantic vector interpolations between two end points. We discuss these in more detail in the section linked above.

In this chapter, we do not attempt to replace any of the above mentioned methods for discovering interesting *points* in latent space. These methods can in fact, be incorporated into the methods that we describe below. Instead, we are interested in granting users methods of control, which allow them to discover and design interesting *trajectories* in latent space. In an image based generative model such as a DCGAN, a single point in latent space produces an image. A *trajectory* in latent space can be thought of as a *sequence* of points, and thus produces a *sequence of images*, a *video*.

Furthermore, the latent space of such a generative model is usually structured so that points which are close to each other in latent space, produce images which are semantically and aesthetically similar to each other. For this reason, a *smooth* trajectory in latent space, will produce videos where images appear to be *morphing* between each other. We discussed this in section 2.1.3: *Latent interpolation*.

An additional outcome of this particular study, is in what we call *trajectory planning*. As we have discussed multiple times throughout this thesis, the generative models that we use, typically have latent distributions that are high-dimensional (e.g. $N_z := 512$) multivariate standard normal $\mathcal{N}(0, 1)$. We know that the mass of such a high-dimensional normal distribution is concentrated in the shell of a hypersphere, often likened to a ‘soap bubble’, with a radius of $R = \sqrt{N_z - 1}$, and a shell ‘thickness’ following a χ distribution with a variance of 1 (J. D. Cook, 2011). As a result, any latent points that are outside of this shell, will produce images that are far from the distribution, and as a result, are likely to be visually undesirable. For this reason, we investigate a number of different methods which ensure that any *trajectory* that we construct, stays within this distribution, i.e. on the surface of the hypersphere. We discuss our methods in subsection 6.4.8: *Trajectory planner details*.

A further point to note, is that in the previous chapter chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we used a *conditional* generative model as the basis for our study. In this chapter, we investigate *unconditional* generative models. In particular, we work with ProGAN (Karras et al., 2017), which is a form of an unconditional

DCGAN. We use ProGAN, as this was the state-of-the-art architecture at the time of our work¹. We have later tested our approach with StyleGAN (Karras et al., 2019)², BigGAN (Brock et al., 2019)³, and StyleGAN2 (Karras et al., 2020), and we have observed that our method works without any modifications.

We choose *unconditional* generative models as the subject for this chapter for a number of reasons. It is much easier to acquire *unlabelled* data to train an unsupervised, unconditional model, than it is to acquire *labelled* data to train a conditional model. Also for this reason, there are already many high quality, pre-trained, unconditional models available for download made available by other researchers. Furthermore, we have already investigated conditional generative models in the previous chapter, and we wish to explore the affordances granted by unconditional models. For more information regarding DCGANs, please refer to subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*, and for unconditional generative models in general, subsection 2.1.1: *Unconditional generative models*.

6.3 Goal and requirements

6.3.1 Main tasks

With all of this information in mind, let us explicitly define our goal and our requirements.

We have a generative model of images, with a high-dimensional space of latent representations (e.g. $N_z := 512$). This could be a model that we have *trained ourselves*, or a *pre-trained* model downloaded from the internet such as BigGAN, or a pre-trained ProGAN/StyleGAN.

The main tasks that we need to address are:

Exploration

We desire that a user should be able to explore and leverage the full scope of the latent space. For this reason, encoding or ‘filtering’ existing images as an input, as we did in the previous chapter chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, is not satisfactory in this case.

Keyframe selection/generation

Our end goal is to devise a solution which allows a user to generate or select a *temporally sparse sequence* of **keyframe images** \mathbf{y}_i from the model, such that image \mathbf{y}_1 corresponds to time t_1 , image \mathbf{y}_2 corresponds to time t_2 , and image \mathbf{y}_i corresponds to time t_i , where the subscript i denotes a sequential integer index, and t_i is the desired point in time that the i th image appears in the video. In other words, we require a solution that allows a user to provide or somehow construct a list of (\mathbf{y}_i, t_i) pairs.

¹section 2.2.7: *Progressive Growing of GANs, ProGAN (2017)*

²section 2.2.7: *StyleGAN (2019), StyleGAN2 (2020)*

³It is worth noting that BigGAN is technically a *conditional* model, as it is conditioned on class labels. However, within each class label, the latent space is still incredibly vast and open for unconditioned exploration. For this reason, we include BigGAN as an example model that we use for this research. We briefly discuss BigGAN in section 2.2.7: *BigGAN (2018)*.

***z*-*y* correspondence**

We would like to work with the generic case of a generative model in which it is not always possible to retrieve a corresponding latent representation \mathbf{z} for an input image \mathbf{x} via an encoder or optimisation as we discussed in section 2.1.3: *Latent representation recovery*. For this reason, our solution should work with (\mathbf{z}_i, t_i) pairs instead of (\mathbf{y}_i, t_i) . These can be thought of as *temporally sparse keyframe z-vectors* which are the latent representations of the (\mathbf{y}_i, t_i) *keyframe images*. However, for us, the user, dealing directly with lists of high-dimensional \mathbf{z} -vectors is not intuitive or even feasible. We prefer to be dealing with images, not high-dimensional \mathbf{z} -vectors. Therefore, our solution should operate with an interface using images and (\mathbf{y}_i, t_i) pairs, while providing the corresponding (\mathbf{z}_i, t_i) pairs to the model in the background.

Trajectory planning

Given that the output will be a video with a fixed framerate (e.g. $fps := 30$), our solution should plot a trajectory through the provided list of temporally sparse (\mathbf{z}_i, t_i) keyframes. If t_i is provided in seconds, the trajectory planner should interpolate and generate intermediate \mathbf{z} points, such that each \mathbf{z}_i is reached in exactly $t_i * fps$ steps. In other words, the trajectory planner produces a new, *temporally dense* and *smooth* list of interpolated \mathbf{z} -vectors, one per frame of video, where the \mathbf{z} -vector at frame $t_i * fps$ is equal to \mathbf{z}_i . The task of the trajectory planner is further complicated by the fact that the mass of the latent distribution is concentrated in the shell of a hypersphere as we have already discussed. Our trajectory planner needs to take this into consideration.

6.3.2 Additional issues

On top of the main tasks mentioned above, we have encountered a number of additional issues. We summarise these below.

Biased distribution

The latent space is not distributed ‘evenly’, or as one might expect or desire. For example, in the case of our test model, we trained on an incredibly diverse set of images — millions of images from over dozens of categories. However, upon inspection of the model’s output, it seems images of *flowers* seem to occupy a very large portion of the model’s latent space. In other words, if we generate a few thousand random samples from the model, we can observe that a disproportionately large percentage of the images generated are of flowers. This is most probably because flowers are overly present in the training data. The inverse is also possible. For example, in the same model, images of bacteria are very sparse. This is most probably because there are not enough images of bacteria present in the training data. In addition, it’s quite common for some areas of the latent space to produce images which are aesthetically and subjectively undesirable, or not suitable for the specific use case we have planned.

In our *exploration* of the latent space, we wish to find ways of taking this into account. One solution which may come to mind to address this, is to readjust the distribution of the training data by adding or removing images, and then retrain the model. However, this is

not always feasible or even possible due to a number of reasons. Some of these models take months to train, so an iterative process of tweaking training data and then retraining is not always feasible. Furthermore, these models often require very large amounts of training data, often tens of thousands, sometimes even millions of images, so removing images can degrade the quality of the model, and adding images is sometimes not possible if more images cannot be found. Finally, many researchers are releasing pre-trained models, without releasing their dataset or even their training code. In our research, we wish to present a solution which can also work with these pre-trained models. Due to these reasons, we look into ways of **biasing or debiasing** the latent space to produce more desirable results.

Perceptually variable speed

As a result of the uneven distribution mentioned above, an additional problem which arises, is that interpolating from latent vector z_A to z_B at a *constant* speed might produce images changing with a *perceptually variable* speed. This can make it difficult to meaningfully control the speed at which the images change when producing a video with narrative. This is especially important when producing videos that accompany music.

Unexpected interpolation outcomes

Finally, it is very difficult to anticipate and imagine the outcomes of trajectories in high dimensions. For example, imagine that the latent representation z_A produces y_A which is an image of a flower, and z_B produces y_B which is an image of a mountain. We may wish to interpolate from z_A to z_B , to produce a video morphing from image y_A to y_B . However, in such high dimensions, this trajectory might unexpectedly pass through point z_C , producing an undesirable output y_C which for example might be an image of a dog.

6.4 System description

Based on the tasks and issues that we mention above, we will now detail our proposed solution.

The first important decision to make when starting a project such as this, is whether we build an entire bespoke tool and User Interface (UI) from scratch, as we have done in the previous four studies, or do we try and leverage existing tools. Both approaches have advantages and disadvantages. Building a bespoke tool from scratch requires considerable investment of time and effort, but eventually provides the most flexibility and power. Trying to leverage existing tools can potentially rapidly accelerate the development of a prototype, at the cost of limited functionality in the long run.

For this particular study, we chose the latter option of leveraging existing tools. We chose this route because our goal is to be able to create videos with precise control over timing. And in many of our use cases, including our collaboration with Max Cooper, it is essential to be able to synchronize specific points in the video with corresponding points in music. Non-Linear Video Editing (NLVE) software have years, even decades, of research and development in this area, with many of the features that we require. For this reason, we chose to leverage and build upon existing NLVE software with the belief that this would save us considerable development

time, and would be the most optimal way of developing a prototype to demonstrate a proof of concept. However, this decision presents us with many new challenges. How can we use a standard NLVE to edit sequences of high-dimensional \mathbf{z} -vectors?

In order to address the different tasks and issues mentioned in the previous section, we have developed a number of different methods and tools, and we have produced many hours worth of carefully constructed films using this method.

Workflow overview

An overview of the entire workflow can be summarized as follows:

1. **Exploration of the model:** bias and de-bias the latent space distribution, create **short journeys** for later editing
2. **The narrative edit:** edit the short journeys using a NLVE to produce *keyframe images*, i.e. (\mathbf{y}_i, t_i) pairs
3. **The narrative conform:** produce the final *keyframe z-vectors*, i.e. (\mathbf{z}_i, t_i) pairs, from the (\mathbf{y}_i, t_i) pairs
4. **Trajectory planning:** produce the final dense, interpolated trajectory \mathbf{Z} from (\mathbf{z}_i, t_i)
5. **The final render:** select a desired model snapshot and produce the final video by rendering \mathbf{Z} through the model

6.4.1 Concepts and definitions

Before we expand on each of these steps, it is important to introduce some concepts which are key to our process.

Conforming an edit

Our solution is analogous — and in fact inspired by — a common practice in video editing and post-production pipelines known as **conforming** an edit. Typically in digital video post-production, working with full-quality, full-resolution video clips is not always feasible due to performance reasons⁴. In this case, one typically performs an edit on a set of **proxy** clips. These are usually lower resolution, lighter-to-decode, easier-to-manage copies of the original videos. Editing with proxies is also known as an *offline* edit (this terminology dates back to working with film and automated machines operating on reels of footage). Once the offline edit is completed on the proxy clips, the edit is *conformed* by applying the same edits to the original, high resolution rushes. To perform this conform process, a file such as an **Edit Decision List (EDL)** is exported from the offline editor. This is an industry standard file format which contains all of the reel and timecode information for each of the video clips, their *in* and *out* points, and their position on the timeline. The EDL is transferred to an *online* system, i.e. with access to the original, full quality rushes, where the final cut is reproduced.

⁴In the days of analogue film editing, where this terminology originates, trying out edits directly on original rushes by cutting them was highly undesirable!

(**z-sequence, video**) pairs

Since both we as users, and NLVEs, are more comfortable working with images, not \mathbf{z} -vectors, we need to associate images with their corresponding \mathbf{z} -vectors. In order to address this, we define (*z-sequence, video*) pairs.

A **z-sequence**, is simply a list of \mathbf{z} -vectors. In other words, it is a *trajectory* in latent space, and we store it on disk as a numpy array. To be more specific, it is a matrix where each row is a \mathbf{z} -vector, the i th row is the vector \mathbf{z}_i .

A **video** is a normal video file — typically an mp4 file with h264 codec — where each frame of the video is the image output from the model decoding the corresponding \mathbf{z} -vector from the corresponding \mathbf{z} -sequence. In other words, the i th frame of the video, is the image \mathbf{y}_i generated by the model decoding \mathbf{z}_i .

In our analogue of **conforming an edit**, these video files act as the easier-to-manage *proxy* clips which we will later load into a NLVE to perform a desired edit. We will then **conform**, by applying this edit to the corresponding \mathbf{z} -sequences, using custom tools that we developed during this research.

Rendering \mathbf{z} -sequences

We generate the videos to be edited using a process we call **rendering**. We appropriate this term from the Computer Graphics literature, where some lower dimensional, custom representation of a scene (for example 3D geometry consisting of vertex, face, material and lighting information) is used to produce a high quality raster image.

In our case, the lower dimensional custom representation is the latent vector sequence. The process of **rendering a \mathbf{z} -sequence**, refers to simply loading the corresponding numpy array, and saving a video where the i th frame of the video is the image \mathbf{y}_i , generated by feeding the row \mathbf{z}_i to the model.

Given one or more \mathbf{z} -sequences, we can *render* them to produce (**z-sequence, video**) pairs⁵. Once we have these pairs, we can then edit the videos in a NLVE as we would edit any normal videos, and then use the tools we developed in this research, to conform the edit on the corresponding numpy arrays containing the \mathbf{z} -vectors.

Snapshots across time

During the training of a generative model, the model’s latent space changes with each training iteration, to hopefully represent the data more efficiently and accurately. However, a noticeable change across these iterations also includes transformations and shifts in space. To give an oversimplified example, what may be an area in latent space dedicated to *mountains* at iteration 70K, might become *flowers* at iteration 80K, while *mountains* slides over to the area previously used to represent *clouds*. This is a vast simplification, as the real effects are little bit more subtle. Nevertheless, there have been no studies conducted on this phenomena, and specifically, on how it may impact creative uses and explorations of generative models.

⁵The association between the numpy array file and video file can be stored in a database, or we choose the much quicker method of simply giving them both the same filename.

Furthermore, when we download a pre-trained model from the internet, often we only have access to the final model provided by the researchers that trained the model. However, when we are training our own models, we can save intermediate **snapshots** at desired intervals. This can be particularly useful while training GANs, because there is no objectively quantifiable point at which we can measurably determine that the training has optimally converged. This is due to the fact that while training, the optimiser is trying to satisfy two opposing objectives, improving both the generator, and the discriminator, by finding a Nash equilibrium of a non-convex minimax game. We discuss this in more detail in subsection 2.3.18: *Deep Convolutional Generative Adversarial Networks (DCGAN)*.

Since there is no objectively quantifiable point at which the training is ‘complete’, a common workaround is to simply set a fixed number of iterations to let the optimiser run for. This is typically in the range [12000..100000], depending on the size and complexity of the training data. In addition, at a fixed interval (e.g. every 1000 iterations) a small selection of random image samples (e.g. 64 samples) are saved to disk. These samples are manually inspected, and if they appear to be subjectively satisfactory and of a desired quality, the training is manually interrupted, and a snapshot is saved and used as the ‘final’ model.

When assessing the random selection of samples, there are a number of factors to consider. First, one looks for satisfactory *visual quality* in the individual images themselves, to ensure that both the coarse structure and fine detail are satisfactory. Second, one looks at the diversity of the images across the selection, to ensure that the model is able to produce images with a satisfactory level of diversity representative of the dataset.

For many typical use cases, where the model will be used only to produce still images, or ‘random walk’ videos, this method of snapshot selection may be satisfactory. However, this method does not allow any way of seeing how the latent space of a snapshot model is organised, how compact or sparse key points are distributed, and what it actually looks like to move from one point to another. For example, looking at the random selection of samples, we may see a perfect reproduction of a flower, or a mountain. We may thus conclude that the model is ready. However, later when we plot a trajectory through this model, we may see that the space between the flower and the mountain has many undesirable artefacts, that maybe was not there a few thousand iterations earlier, or maybe will disappear a few thousand iterations later.

For this reason, we believe that this snapshot selection method is not ideal, particularly when the goal is to design narratives and construct trajectories, where we desire a high level of meaningful control.

An ideal solution would allow us, when we are designing our trajectories, to visualize the results of those trajectories, in not just a single snapshot, but in many snapshots simultaneously, covering a wide range of iterations.

To achieve this, we save snapshots at regular intervals, and render z -sequences from multiple snapshots. When we render a z -sequence, instead of rendering it from just one model, we render it from multiple snapshots. We then tile the results of each of these renders together into a single video. In other words, the rendered video of a (z -sequence, video) pair, is no longer a video rendered from a *single* model, but instead, each frame of the video contains the output from *multiple snapshots* arranged in a grid. This can be seen clearer in Fig. 6.2.

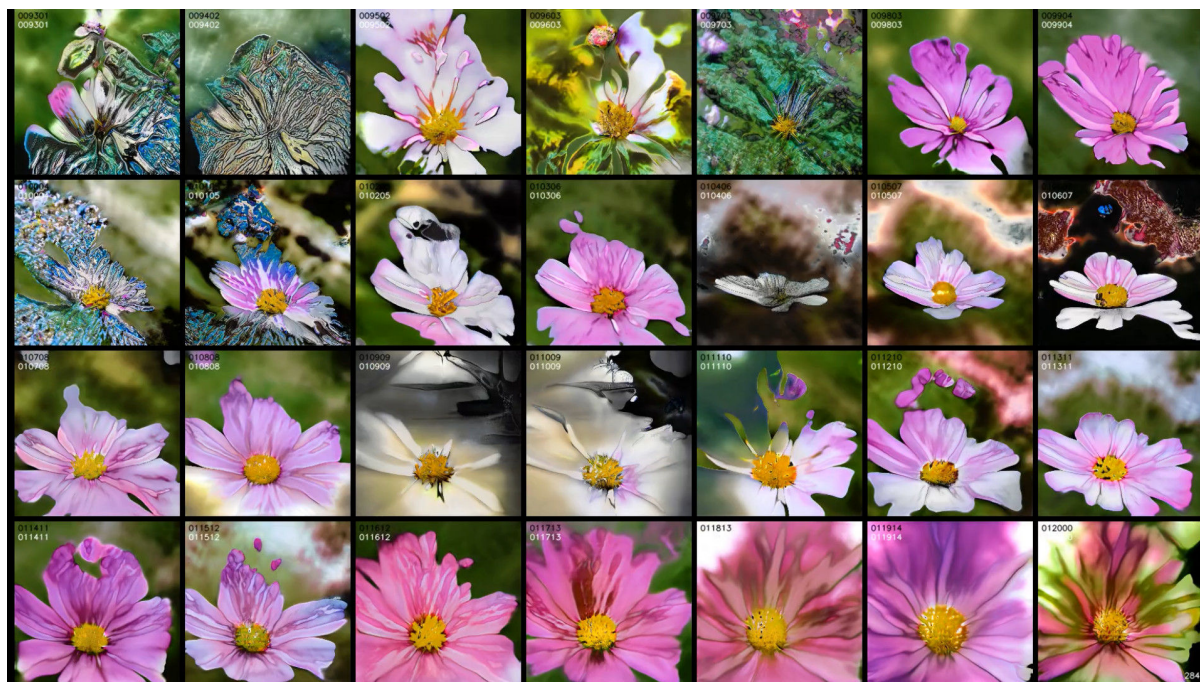


Figure 6.2: An example frame from a rendered video of a (z -sequence, video) pair. Each of the 28 tiles in this image, shows the image produced by decoding the same z -vector from 28 different snapshots of the same model where each snapshot is separated by 100 iterations. The snapshot iteration number is also included in each square panel for reference, although in a font perhaps too small for print.

Rendering multiple snapshots in a grid on a single frame in this way gives us an overview of how the latent space has evolved across training iterations.

Furthermore, we do not do this simply to observe how the model’s latent space evolves over time. This method also allows us to easily see and select the most *aesthetically desirable snapshot(s)* for the *particular trajectory that we have designed*.

In other words, instead of *first* picking a snapshot as the final model, as is the current typical workflow, and *then* creating a trajectory, we explore trajectories in all of the snapshots simultaneously. Throughout the entire process that we outline below, we work with these *tiled* videos. This allows us at every step, to visualise how the trajectory looks like from each of these different snapshots. At the very end, once we have produced our final draft tiled video, we can select the particular snapshot which provides us the most aesthetically desirable results, and render the same trajectory in high resolution with only that snapshot.

The selection of snapshots to include is based on a simple subjective assessment of the visual quality of the snapshots. In our specific test cases, we train a number of ProGAN models for a fixed number of iterations, namely 12000, as suggested by the authors of the paper. Looking at the random image samples generated during training, we noticed that the quality of the outputs started to become satisfactory at around 10,000 iterations. Based on this observation, we decide to save snapshots every 100 iterations, and include the final 28 snapshots in the render. This covers iterations 9,300 through to 12,000, and gives us ample options to choose from. In addition, the 28 images can be tiled into a 7x4 grid. This makes viewing the results on a typical 16:9 monitor very practical.

6.4.2 Exploration of the model

Exploration is the first step that we outline in section 6.4: *Workflow overview*, where we **bias and de-bias** the latent space distribution, and create **short journeys** for later editing.

If we already have a strong idea of the narrative that we would like to construct, and we are already familiar with the particular model that we are working with, we can keep this exploration phase very brief and jump straight to the *narrative edit*. However, if we are dealing with a new model, and we do not yet know what the model is capable of, this phase can provide very valuable insights.

The exploration phase can be expanded as follows:

1. We take many — e.g. one thousand — unbiased and independent samples from the model’s latent distribution $\mathcal{N}(0, 1)$. We render and save this as a (z -sequence, video) pair, where each frame is an entirely different random image. Stepping through this video frame by frame gives us an idea of what the model has learnt, and how the latent space is distributed. It also gives us an idea of how the distribution changes across subsequent training iterations, and which snapshots provide more aesthetically desirable images.
2. We edit this video in a NLVE to remove any undesirable images, and to bias or de-bias the distribution. For example, to address the issue with flowers being overly present in our model, we simply remove a small random selection of flowers from the video. To address the issue with there not being enough images of bacteria, we edit the video to duplicate frames containing bacteria. We continue performing simple edits such as this until we are content with the overall distribution of the different types of images in the video. At this stage we are not performing edits to construct a narrative or to tell a story. We are simply removing or duplicating frames to bias the distribution of different types of images.
3. Once we are happy with the distribution of images in the edit, we run our script to conform the edit on the original z -sequence. This produces a new z -sequence — which we refer to as a z_{dist} -sequence — where each frame is still an entirely different random image, but which has a more desired distribution, i.e. that contains no undesirable images, and a satisfactory balance between different images types.
4. We render the conformed z_{dist} -sequence to a new video, and we inspect this video in the NLVE. If necessary, we can repeat steps 2–3 to further fine-tune the distribution until we are entirely satisfied with it. However, we have not found this to be necessary.
5. Optionally, before rendering the conformed z_{dist} -sequence, we can add a small amount of Gaussian noise to it, for example $\mathcal{N}(0, 0.01 - 0.05)$. This allows us to explore the neighbourhoods of the currently selected frames — for example, to include and investigate more images of bacteria, with more variation.

The steps above produce a final (z_{dist} -sequence, video) pair, where each frame is a random image, with desired distribution of images and no undesired images.

Short journeys Once we are satisfied with the distribution of the random samples, we generate and render hundreds of random **short journeys**. We do this by loading the final z_{dist} -sequence in python, and simply select random pairs or triplets of z -vectors, and we render interpolations between them. This can be seen in Alg. 3.

Algorithm 3: Deep Meditations: Generation of random *short journeys*

```

1 snapshots  $\leftarrow$  Load(model snapshots);
2  $z_{\text{dist}}$ -sequence  $\leftarrow$  Load( $z_{\text{dist}}$ -sequence numpy array);

3 for  $i \leftarrow 1$  to num_desired_short_journeys do
4      $N \leftarrow$  RandomSelect([2, 3]); // Randomly select 2 or 3
5      $z_{\text{keyframes}} \leftarrow$  RandomSelect( $z_{\text{dist}}$ -sequence,  $N$ ); // Select  $N$  random items from
         $z_{\text{dist}}$ -sequence
6      $t \leftarrow$  desired time interval between keyframes, e.g. 1–10 seconds;
7      $z_{\text{dense}} \leftarrow$  TrajectoryPlanner( $z_{\text{keyframes}}$ ,  $t$ );
8     Render(snapshots,  $z_{\text{dense}}$ );
9 end

```

This produces hundreds of short (z -sequence, video) pairs that contain smooth, slow interpolations between two or three keyframes, where the keyframes are chosen from our preferred distribution z_{dist} . Viewing these videos gives us an idea on how the model transitions between selected desired images. For example, using such a method, we can quickly see that the shortest path between an image of a mountain and an image of a flower might travel through the latent space region of buildings, and this may not be desirable. We can use this as a cue to insert an additional keyframe in-between.

We repeat the previous step, honing in on the short journeys which seem promising. Optionally, we can again add varying amounts of Gaussian noise to the z -vectors to explore the neighbourhoods of selected frames and journeys.

6.4.3 The narrative edit

The previous steps produce an arsenal of short snippets of (z -sequence, video) pairs, as well as a single long (z_{dist} -sequence, video) pair with many desirable images.

To perform the **narrative edit**, we simply load the *videos* that we like, or that contain images or sections that we like, into the NLVE, and we edit them together. In other words, we select frames or short sequences from these videos, and we sequence them and lay them out on the NLVE timeline with the desired timing, synchronizing to audio if necessary.

During this edit, we essentially lay down the *keyframe images* y_i , i.e. *temporally sparse destination points*. When we place video frames onto the timeline, we have two options. The first option is to *leave gaps* between the keyframes, placing only images and video frames at specific points in time that we deem significant. The gaps will later be filled in by our *trajectory planner*, which interpolates between neighbouring keyframe images. Due to the smooth nature of the model’s latent space, this interpolation manifests itself visually as a kind of *morph* between the keyframes, as we discussed in section 2.1.3: *Latent interpolation*. The second option, is

6.4. SYSTEM DESCRIPTION

to leave *no gaps* between keyframes. In this case however, upon loading the z -sequence in the *trajectory planner*, we can specify a fixed time interval, for example 5 seconds, and the keyframes will be treated as if they are 5 seconds apart.

As a result of this edit, the NLVE project contains information regarding *keyframe images*, i.e. (y_i, t_i) pairs.

As a side note, for our NLVE software, we chose the open-source **Kdenlive** for the GNU/Linux platform. One of the reasons for this, is because we are working solely on GNU/Linux. However, our solution does not involve any modifications or plugins to the software itself, but rather it is a workflow involving a number of stand-alone tools, which could be adapted to other NLVEs. This means that the same workflow that we describe in this section, could be applied to other NLVEs which support any kind of easy-to-parse project file.

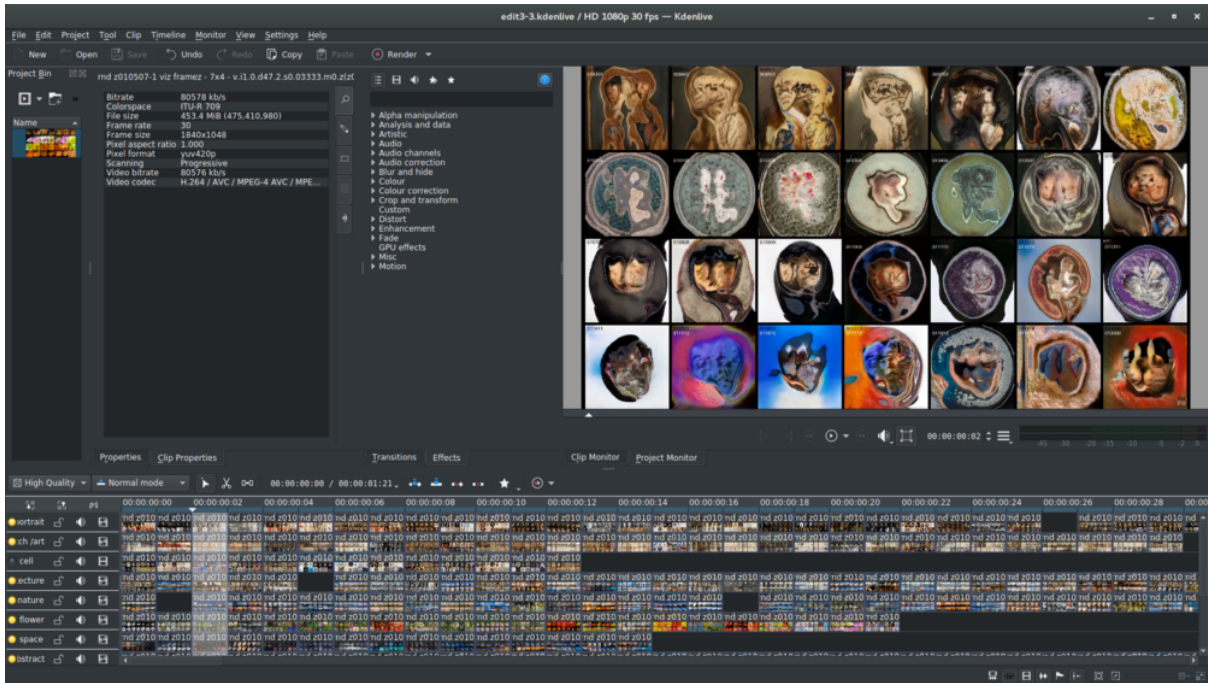


Figure 6.3: A screenshot of editing z -sequence videos in Kdenlive with tiled snapshots.

6.4.4 The narrative conform

Having constructed a list of (y_i, t_i) keyframe images during the *narrative edit*, the **narrative conform** is simply a process of running our *conform script* on the *narrative edit NLVE project file*. This produces a list of temporally sparse *keyframe z-vectors*, i.e. a numpy matrix of (z_i, t_i) pairs.

Unfortunately, Kdenlive does not support EDL⁶. However, the project file format used by Kdenlive is itself a text based XML with open specifications. This allows us to easily parse the project file in our own python based tools. One of the open-source tools⁷ that we developed during this research, is a parser which conforms the edit as shown in Alg. 4. With relatively

⁶EDL is *Edit Decision List*, we discuss this in section 6.4.1: *Conforming an edit*

⁷<https://github.com/memo/py-msa-kdenlive>

minor modifications to this tool, EDL support could be implemented, which would permit the use of other NLVEs such as Adobe Premiere, Apple Final Cut, and Avid Media Composer.

Algorithm 4: Deep Meditations: *Conforming* the edit

- 1 Load Kdenlive project XML file;
 - 2 Inspect the edits and retrieve the names for each of the video clips;
 - 3 Load the corresponding numpy \mathbf{z} -sequences for each of the video clips;
 - 4 *Conform the edit* by splicing the numpy arrays in the same manner as the videos;
 - 5 Export a new \mathbf{z} -sequence to disk;
-

As mentioned, the output of this conform process is a list of temporally sparse *keyframe z-vectors*. This is a numpy matrix of (\mathbf{z}_i, t_i) pairs, which is saved to disk, and can then be sent to the *trajectory planner* for interpolation.

6.4.5 Trajectory planner

The **trajectory planner** takes the temporally sparse (\mathbf{z}_i, t_i) *keyframe z-vectors* produced by the *conform*, and produces the final dense trajectory \mathbf{Z} which will eventually be rendered by the model.

As we have mentioned before, the latent distribution of our models are concentrated in the shell of a hypersphere. For this reason, we need to ensure that the trajectory planner takes this into account. This is a rather complex issue that we have invested considerable time into. Instead of detailing all of our findings here, we have devoted a separate section to it below in subsection 6.4.8: *Trajectory planner details*.

6.4.6 The final render

Rendering the final dense \mathbf{z} -sequence \mathbf{Z} produced by the *trajectory planner*, produces a final *tiled* video. In this video, we can see the same trajectory rendered from each of the available snapshots, just as we saw in Fig. 6.2. Viewing this video, we can see which of the snapshot(s) provides the most aesthetically pleasing results, and we can then re-render the video in full resolution with only one snapshot, or a smaller selection of snapshots.

6.4.7 Model architecture and data

We applied the approach mentioned in this paper on a number of different models and architectures, however the primary test case we refer to (and from which we also show the results) is a ProGAN trained on over 100,000 images scraped from the photo sharing website Flickr. The dataset is very diverse and includes images tagged on Flickr with: art, cosmos, everything, faith, flower, god, landscape, life, love, micro, macro, bacteria, mountains, nature, nebula, galaxy, ritual, sky, underwater, marinelife, waves, ocean, worship and more. We include three thousand images from each category and train the network with no classification labels. Given such a diverse dataset without any labels, the network is forced to try and organize its distribution based purely on aesthetics, without any semantic information. Thus in this high-dimensional latent space we find directions allowing us to seamlessly morph from swarms of bacteria to clouds of

nebula, oceanic waves to mountains, flowers to sunsets, blood cells to technical illustrations etc. Most interestingly, we can perform these transformations across categories while maintaining overall composition and form.

6.4.8 Trajectory planner details

In this section, we detail some of our investigations into **trajectory planning**. As we have mentioned, the *trajectory planner* takes the temporally sparse (z_i, t_i) *keyframe z-vectors* produced by the *conform*, and produces the final dense trajectory \mathbf{Z} which will eventually be rendered by the model. In order to do this, the aim is to produce the final dense trajectory \mathbf{Z} , such that at time t_i , the trajectory goes through the point z_i , and for every frame in between, z -vectors are smoothly interpolated.

One of the challenges that we face, is that our latent distribution is a high-dimensional standard normal $\mathcal{N}(0, 1)$. As a result, the mass of the distribution is concentrated in the shell of a hypersphere with a radius $R = \sqrt{N_z - 1}$, and a shell ‘thickness’ following a χ distribution with a variance of 1 (J. D. Cook, 2011). For this reason, the trajectory planner needs to ensure that any z that it generates, needs to be on, or very close to, the surface of this hypersphere. If this criteria is not met, the images produced will be far from the distribution, and potentially very undesirable. We investigate a number of different methods to address this.

Spherical spatial interpolation, linear temporal interpolation

Given just two points, z_a and z_b , if we linearly interpolate between them, this will diverge from the distribution. This can be visualized as a straight line between two points on the surface of a sphere, diverging from the surface of the sphere. As recommended by (J. D. Cook, 2011; White, 2016a) and many others, spherical interpolation provides the most desirable results in this case. However, when we have more than two points, using spherical interpolation becomes problematic.

The first approach that might come to mind, is to use spherical interpolation between each successive two points, z_i and z_{i+1} . An overview of this can be seen in Alg. 5. However, using this method, we observe that this produces visibly noticeable discontinuities *in the movement* of the output video, due to sudden changes in speed and direction in the latent space. Even though we are using *spherical* and not *linear* interpolation, this simply refers to the interpolation in *space*. However, temporally, the interpolation remains linear. Any experienced animator will know that linear interpolation across time often causes temporal discontinuities at the keyframes due to sudden changes in speed.

These artefacts can be seen in Fig. 6.4. For this test, we generate random samples from a 512 dimensional standard normal distribution $\mathcal{N}(0, 1)$, which can be thought of as points that lie on or very near the surface of a hypersphere with radius $R = \sqrt{512 - 1} = 22.6$. We interpret these random samples as keyframes, placed one second apart, and we use spherical interpolation between each successive two points as outlined in Alg. 5, to produce a dense trajectory \mathbf{Z} . In this case we consider a video frame rate of 15 frames-per-second, which means we interpolate between each successive keyframe in 15 equal steps.

Algorithm 5: Deep Meditations: Produce a dense trajectory Z using *spherical interpolation*

```

// Inputs
1  $z_i \leftarrow [z_1, z_2, \dots, z_n]$ ; // Array of  $z$ -vector keyframes
2  $t_i \leftarrow [t_1, t_2, \dots, t_n]$ ; // Array of corresponding timestamps (in seconds) for  $z_i$ 
3  $\text{fps} \leftarrow 15$ ; // Desired frame rate
4  $f(z_A, z_B, \text{perc})$ ; // Spherical interpolation function

// Variables
5  $\text{nkeyframes} \leftarrow \text{Length}(z_i)$ ; // Number of keyframes
6  $Z \leftarrow []$ ; // Initialize empty array to store interpolated  $z$ -vectors

// Loop over all keyframes
7 for  $i \leftarrow 2$  to  $\text{nkeyframes}$  do
8   duration  $\leftarrow t_i - t_{i-1}$ ; // Duration (in seconds) between adjacent keyframes
9   nsteps  $\leftarrow \text{duration} * \text{fps}$ ; // Number of steps between keyframes

   // Interpolate between adjacent keyframes
10  for step  $\leftarrow 1$  to nsteps do
11    perc  $\leftarrow (\text{step} - 1) / \text{nsteps}$ ; // Interpolation percentage 0...1
12     $z_{\text{cur}} \leftarrow f(z_{i-1}, z_i, \text{perc})$ ; // Apply spherical interpolation to calculate
        current interpolated  $z_{\text{cur}}$ 
13     $Z.\text{append}(z_{\text{cur}})$ ; // Append to dense  $Z$  array
14  end
15 end
16 return  $Z$ 

```

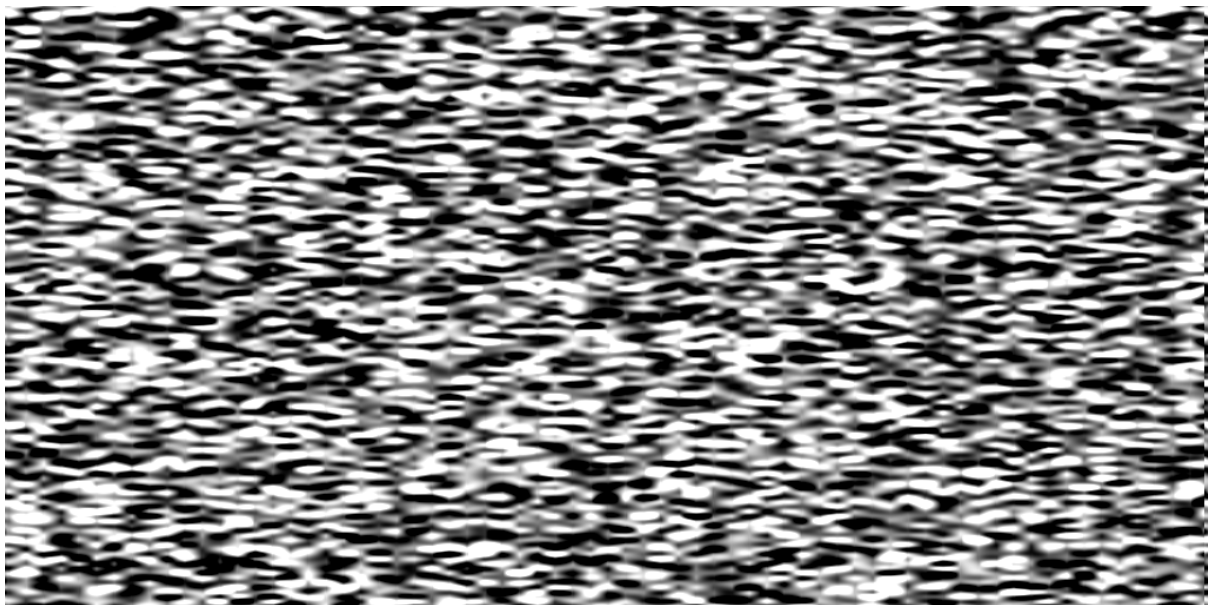


Figure 6.4: An interpolated, dense Z sequence produced using spherical interpolation. Time flows left to right. Upon close inspection, one can notice vertical notches. These indicate discontinuities every time a keyframe is reached, and a new target keyframe is selected.

In Fig. 6.4, a single pixel wide vertical slice represents a single \mathbf{z} -vector, and time flows left to right. Because we placed the keyframes one second apart, with a framerate of 15, in this image, the keyframes occur 15 pixels apart. Upon close inspection of the image, we can see notch-like vertical artifacts every 15 pixels. This happens when the spherical interpolation reaches its destination \mathbf{z} -keyframe, and a new target \mathbf{z} -keyframe is set. In other words, the inner loop on line 10 in Alg. 5 is reached. This creates a sudden change in speed and direction.

Spherical spatial interpolation, easing temporal interpolation

A simple remedy that may come to mind, is to smooth the movement over *time*, using what are known in the field of animation as *easing* functions, and more formally known as *sigmoid* functions. These are functions that create a *linear* \mapsto *S* shaped mapping, and for this reason they are also known as *S-curves*.

A very common easing function is the cubic $S(t) = 3t^2 - 2t^3$, where t is limited to the range $(0, 1)$. We can see that $S(0) = 0$ and $S(1) = 1$. Furthermore, if we take the derivate of $S(t)$ we find that $S'(t) = 6t - 6t^2$, and both $S'(0) = 0$ and $S'(1) = 0$. In other words, this cubic *S-curve* goes through the points $(0, 0)$ and $(1, 1)$ while having a zero first derivative at both end points. Using an easing function such as this, while interpolating between adjacent keyframes, ensures that there are no first order discontinuities, by ensuring that velocity is zero at the keyframes. In some sensitive cases however, using even cubic easing is not sufficient, as this can produce *second order* discontinuity. For this reason a quintic easing function of $S(t) = 6t^5 - 15t^4 + 10t^3$ can be used where the second derivative — i.e. *acceleration* — is also zero at both $t = 0$ and $t = 1$ (Perlin, 2002).

We can use these, or other similar functions on the *perc* parameter in Alg. 5 while performing the spherical interpolation in line 12. While this does remove the sudden changes in speed and direction, it does so in an undesirable manner. We see that the whole video momentarily grinds to a halt at every keyframe. This should not come as a surprise, as that is exactly what we are doing by using an easing function *only on time* with zero derivatives at the end points. It becomes clear, that in order to address the issue in a desirable manner, we have to smooth the transitions at keyframes *across space* as well as time, ideally without dropping speed down to zero.

Gaussian filter with reprojection

The first method we investigate to smooth transitions across space and time, involves applying a Gaussian filter to the \mathbf{Z} trajectory produced in section 6.4.8: *Spherical spatial interpolation, linear temporal interpolation*. \mathbf{Z} can be thought of as matrix of N_z columns, where N_z is the dimensionality of the model’s latent distribution. Each row is an interpolated \mathbf{z} -vector, and the number of rows is equal to the number of frames in the final video to be rendered.

We apply a one dimensional Gaussian convolution vertically across this matrix, in other words, only across the *time* dimension. This filters each dimension of the motion trajectory individually, without filtering across the components of the \mathbf{z} positions. This does nicely eliminate any sudden changes in speed and direction. However, it also pulls the \mathbf{z} positions towards the center of the latent space, away from the surface of the hypersphere. Since the trajectory

diverges from the distribution, this has the potential to produce undesirable images. To alleviate this, we normalize the filtered \mathbf{Z} matrix and reproject all of the points back onto the surface of the hypersphere. This can be seen in eqn. (6.1)

$$\mathbf{z} \leftarrow R \frac{\mathbf{g}(\mathbf{z}, \boldsymbol{\sigma})}{\|\mathbf{g}(\mathbf{z}, \boldsymbol{\sigma})\|} \quad (6.1)$$

where $\mathbf{g}(\mathbf{x}, \boldsymbol{\sigma})$ is the Gaussian filter function with standard deviation $\boldsymbol{\sigma}$, and $R = \sqrt{N_z - 1}$ is the radius of the hypersphere.

The standard deviation of the kernel depends entirely on subjective aesthetic preferences, and certain characteristics of the video such as the general pace of the video and spacing of keyframes. Unsurprisingly, a larger kernel provides slower and smoother movement, while a smaller kernel provides less smoothing.

Projecting the filtered \mathbf{Z} trajectory back onto the hypersphere in this manner does bring the points back into the distribution, and produces more aesthetic and desirable images. However, depending on the nature of the trajectory, the final smoothed and reprojected trajectory might not travel exactly through the desired keyframes \mathbf{z}_i , but instead might pass nearby. This is particularly likely to happen if the kernel size is quite large. This is not always a major problem. Due to the nature of how the model’s latent space is organized, points which are close in latent space, produce images which are semantically and aesthetically similar. So missing a keyframe by a *small* amount, will produce an image which is *very similar* to the desired keyframe image. Thus, selecting an optimal kernel size becomes a balancing act between selecting a large enough size such that the movement is adequately smoothed, but not too large so as to minimize the divergence between the trajectory and desired keyframes. We find that an optimal kernel size to be in the order of half of the spacing of the keyframes. With a kernel size in this range, the movement is smoothed enough so that keyframes are not noticeable when viewing the video, while simultaneously the trajectory goes close enough to the keyframes so that the images produced are very similar to the desired keyframes.

To summarize, this method creates a perfectly smoothed trajectory, however it fails to hit the desired keyframe *positions* perfectly. For most of the works that we created for this chapter, we used this method of interpolation, as precise position turned out to be not very significant as long as the trajectory passed close enough to the keyframes to be semantically and aesthetically similar.

Physical Interpolation

Another method we investigate, is a physics based dynamical system. In the high-dimensional latent space, we create a dynamical particle connected to the next keyframe \mathbf{z} position with a heavily damped spring, with zero length. This ensures that the particle always moves towards the next keyframe, without any discontinuities in speed or direction. We also connect the particle to the origin, i.e. all components of the vector equal to zero, with a damped spring with length $\sqrt{512 - 1} = 22.6$. This ensures that the particle always stays close to the surface of the hypersphere, i.e. the distribution. Finally, we add a maximum speed cap to the particle,

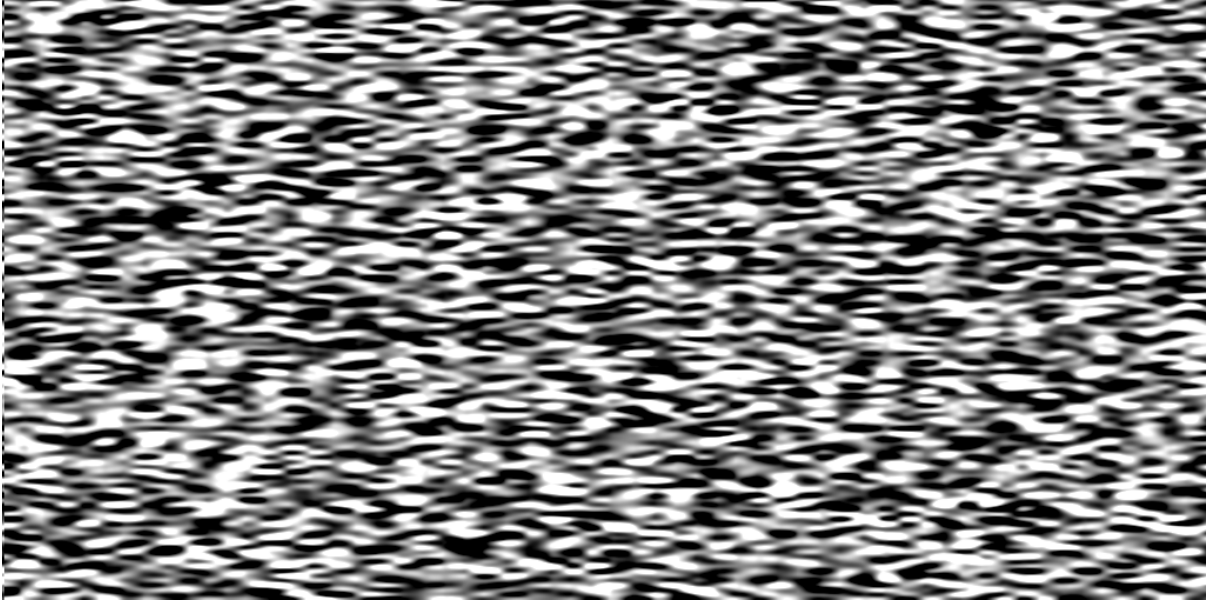


Figure 6.5: An interpolated, dense \mathbf{Z} sequence produced using physical interpolation. Time flows left to right. The vertical notches that were visible in Fig. 6.4 are no longer an issue.

to prevent excessive speeds. An overview of the method can be seen in Alg. 6.

A visualization of a dense \mathbf{Z} trajectory produced using this method can be seen in Fig. 6.5. Note that the vertical notches have disappeared. This is indicative of the transitions between keyframes being much smoother. In other words, upon reaching a particular keyframe, there are no sudden changes in speed or direction. Instead, the interpolated \mathbf{z} vector, gradually changes speed and direction before heading over to the next \mathbf{z} -keyframe, while simultaneously trying to stay within the shell of the hypersphere. This is also noticeable while watching the rendered video, it is almost impossible to notice when the keyframes are occurring, and the overall movement is incredibly smooth.

It is difficult to distinguish which method produces *smoother* and more aesthetically pleasing results between this method, and the previously mentioned section 16: *Gaussian filter with reprojection* method. However one advantage that this method has over the Gaussian reprojection method, is that the trajectory is guaranteed to travel through all of the \mathbf{z} -keyframes with the physical interpolation.

However, this comes at the cost of a lack of precise *timing*. The behaviour of this method is highly sensitive to the *dynamics parameters*, particularly the spring *stiffness* and *damping*. While it is relatively trivial to adjust these parameters to achieve aesthetically pleasing results, it is less trivial to adjust these parameters to achieve aesthetically pleasing results while simultaneously hitting the desired keyframes \mathbf{z}_i at exactly the desired times t_i . For this reason, we use this method when very precise timing — such as synchronizing to music — is not necessary.

In the following section we discuss a couple of other potential approaches such as a modification to *de Casteljaou's algorithm* and using *differential geometry*.

Algorithm 6: Deep Meditations: Produce a dense trajectory Z using *physical interpolation*

```
// Inputs
1  $z_i \leftarrow [z_1, z_2, \dots, z_n]$  // Array of  $z$ -vector keyframes
2 SpringForce (pos0, pos1, spring stiffness, rest length);

// Dynamics Parameters
3 max_speed  $\leftarrow$  maximum speed of particle in latent space;  $k_t \leftarrow$  spring stiffness
   connecting to target  $z$ ;
4  $k_o \leftarrow$  spring stiffness connecting to origin;
5 damping  $\leftarrow$  higher values provide more damping, less momentum;
6 distance_threshold  $\leftarrow$  distance to target keyframe  $z$  before switching to next keyframe  $z$ ;

// Variables
7  $N_z \leftarrow \text{Length}(z_0)$ ; // number of dimensions, e.g. 512
8  $R \leftarrow \text{Sqrt}(N_z - 1)$ ; // radius of hypersphere
9  $v \leftarrow N_z(0)$  // Initialize velocity vector to zero-vector of dimensions  $N_z$ 
10  $z_{\text{cur}} \leftarrow z_1$ ; // Current  $z$  position, start at first keyframe
11 nkeyframes  $\leftarrow \text{Length}(z_i)$ ; // Number of keyframes
12  $Z \leftarrow []$ ; // Initialize empty array to store interpolated  $z$ -vectors

// Loop over all keyframes
13 for  $i \leftarrow 2$  to nkeyframes do
14   while  $\|z_i - z_{i-1}\| < \text{distance\_threshold}$  do
15      $v *= \text{damping}$ ; // Apply damping
16      $v += \text{SpringForce}(z_{\text{cur}}, z_i, k_t, 0)$ ; // Spring force to target keyframe
17      $v += \text{SpringForce}(z_{\text{cur}}, 0, k_o, R)$ ; // Spring force to origin

     // Cap speed of particle
18     if  $\|v\| > \text{max\_speed}$  then
19        $v = \text{max\_speed} * v / \|v\|$ 
20     end

21      $z_{\text{cur}} += v$ ; // Add velocity to current position
22      $Z.append(z_{\text{cur}})$ ; // Append to dense  $Z$  array
23   end
24 end
25 return  $Z$ 
```

6.5 Conclusion

We present this research as a first step in many, towards enabling users to meaningfully explore and control journeys in high-dimensional latent spaces to construct stories, using and building upon industry standard tools and methods with which they may already be comfortable. Our ultimate goal is to enable users to creatively express themselves, and meaningfully control the way in which they produce time-based media using deep generative models as a medium.

At the time of our research, there were no methods or tools available allowing a user *any* form of control — let alone what we consider *Meaningful Human Control* — over the narrative of a video generated with an image based Deep Neural Network. As a first step in this direction, the method that we present here works. Using this method, we have produced many videos of varying lengths, ranging from very short (one minute), to almost feature length (one hour). We have exhibited these works in different contexts including a music video and live concert visuals for the electronic musician Max Cooper, and an immersive, architectural audio-video installation in a hotel lobby designed by Foster & associates.

However, many aspects of our process can be improved; especially from a user experience point of view. We chose to leverage the functionality of existing tools. Specifically, we chose to build our workflow around an existing NLVE, and this allowed us to prototype the desired functionality relatively quickly, since the NLVE’s user interface has been designed and developed over decades and optimised for editing videos in a non-linear fashion, and synchronizing to audio if need be. However, the rest of our workflow consists of a number of python scripts, and executing various different cells of an iPython notebook. For this reason, it is not very user friendly. As a next step, we would highly recommend incorporating the functionalities that we mention above, into a stand-alone application where these functionalities can be triggered and controlled via a single GUI.

In addition, the exploration of the model’s latent space is currently a bit cumbersome, as it requires switching back and forth between the NLVE — where the viewing and editing takes place, and the python scripts — where the conform and rendering takes place. Collating all of this functionality into a single custom GUI would again allow for a more streamlined, realtime experience for the user.

Finally, we have discovered that *trajectory planning* is a very important issue if quality of both *movement* and *image* is to be preserved when creating videos using deep generative models. We already know that using *linear interpolation* between successive keyframes causes the trajectory to diverge from the distribution, resulting in undesirable images. In addition, we have tested and observed that using *spherical interpolation*, results in noticeable discontinuities in movement, which we find very undesirable.

We have implemented and tested two new methods, and both *physical interpolation* and *Gaussian reprojection interpolation* provides very nice smooth trajectories with no noticeable discontinuities at all. Physical interpolation is able to produce trajectories that travel *precisely* through all desired *z*-keyframe positions, but *not* at precisely the desired *times*. Gaussian reprojection interpolation is able to produce trajectories that travel *close* to all desired *z*-keyframe positions, at *precisely* the desired *times*. We call this the *Heisenberg Uncertainty Principle of latent space navigation*.

It's worth adding, that if one were determined to have a smooth trajectory that both travels through all desired z -keyframe positions at precisely the desired times, a dual pass approach we have used is to use the physical interpolation method and render a final video, and then time remap the video in a software such as Adobe AfterEffects or Adobe Premiere. However, we have rarely found this to be necessary, as the position precision of Gaussian reprojection has generally been sufficient for our needs.

A more mathematically rigorous approach one could try is based on *de Casteljau's algorithm*. De Casteljau's algorithm is a recursive method typically used to construct polynomial algebraic curves (known as *de Casteljau Bezier* curves) from a small number of sparse data points (known as *control points*) which define the overall shape. The method involves iteratively calculating affine linear combinations of successive sparse points to create the final smooth curve. In our case, instead of using *linear* interpolations between successive points at each iteration, one could try using a similar recursive approach with *spherical* interpolation between successive points where the initial control points are the z keyframes. This should create a spatially and temporally smooth polynomial curve on (or near) the surface of the hypersphere. However, since the points of a de Casteljau Bezier curve does not go through all of the control points, this is also likely to not travel exactly through all of the desired z keyframes. For this reason, a Hermite/Catmull-Rom based approach projected onto the hypersphere might yield more desirable results.

Another method that we believe could work — particularly in conjunction with Hermite/Catmull-Rom splines as discussed above — is using *differential geometry and Riemannian manifolds* (Miolane et al., 2018). With this approach we can project an offset vector — such as velocity — onto the tangent space of a manifold, which in our case is a hypersphere. From there, we can ensure that all transformations take place on the surface of the hypersphere directly. Our early investigations in this direction look promising, however, we do not yet have conclusive results, and we recommend this as a direction to study. One of the advantages of this approach is that it could potentially be easily adapted to run in realtime.

Chapter 7

Conclusion

A typewriter? — why shd it only make use of the tips of the fingers as contact points of flowing multi directional creativity. If I invented a word placing machine, an “expression-scriber”, if you will, then I would have a kind of instrument into which I could step & sit or sprawl or hang & use not only my fingers to make words express feelings but elbows, feet, head, behind, and all the sounds I wanted, screams, grunts, taps, itches, I’d have magnetically recorded, at the same time, & translated into word—or perhaps even the final xpressed thought/feeling wd not be merely word or sheet, but itself, the xpression, three dimensional—able to be touched, or tasted or felt, or entered, or heard or carried like a speaking singing constantly communicating charm. A typewriter is corny!! — Amiri Baraka (Baraka, 1969)

7.1 Summary of research background and objectives

At the time that we started our research, **Creative DL** — the field investigating the application of DL to the production of artistic and creative works — was very much in its infancy, and was just starting to show potential (Sutskever et al., 2011; Boulanger-Lewandowski et al., 2012; Sutskever, 2013; Graves, 2013; Nayebi & Vitelli, 2015; Sturm, 2015; Ha, 2015; Gregor et al., 2015; Nguyen et al., 2015; Gatys et al., 2015a, 2015b; Mordvintsev et al., 2015; Radford et al., 2015; Nayebi & Vitelli, 2015). While these methods demonstrated great possibilities, they allowed for very little, if any, *Meaningful Human Control*, and no *Realtime Continuous Control* whatsoever. Furthermore, the outputs generated by these methods were of relatively low quality, and the methods themselves were not necessarily very stable. As a result, the majority of research in this area was focused more on improving the *quality* and *stability* of such methods, without necessarily thinking about Meaningful Human Control, or Realtime Continuous Control.

At this time, we hypothesised that both Meaningful Human Control and Realtime Continuous Control would become very important areas of research within Creative DL, and were vastly under-explored. Operating under the expectation that these methods would be improved in terms of quality and stability — especially with massive investments from industry giants such as Google, Microsoft, Facebook, Adobe, OpenAI, and Nvidia — we chose to focus our research specifically on the topic of Realtime Continuous Meaningful Human Control.

We set out to investigate **Deep Visual Instruments**: realtime interactive generative systems that exploit and leverage the capabilities of state-of-the-art Deep Learning algorithms, while allowing *Meaningful Human Control* over the generated media, in a *Realtime Continuous* manner. Our aim was to **explore new modes of performative, artistic expression, using Deep Learning models as a medium and visual instrument**.

We are very pleased to observe that over the years, Creative DL with *Meaningful Human Control* has become a rapidly growing, very exciting area of research (Isola et al., 2016; Zhu et al., 2017; Ha & Eck, 2017; Karras et al., 2017; Park et al., 2019; Karras et al., 2019; Simon, 2019; Bau et al., 2019; Karras et al., 2020; Härkönen et al., 2020; Jiang et al., 2020; Broad et al., 2020), and we are very pleased to have been part of this movement. However, we believe that *Realtime Continuous Control* within Creative DL is still vastly under-explored. We hope and believe, that this too will become an active area of research in the upcoming years, and we hope that our work will be of value.

In this thesis, we first define the two key criteria that we frame our research around, and that we believe to be incredibly valuable, and yet a major gap in current Creative DL research, both from the artistic side, and technical side.

In section 1.3: *Meaningful Human Control*, we first define what we mean by **Meaningful Human Control**, a term that we reappropriate from the *Autonomous Weapons Systems* literature. We provide examples of types of interactivity that we consider do *not* qualify as Meaningful Human Control. For example, ‘pressing a button’ that triggers some autonomous process to produce some creative outputs, with no further input from a user. Or adjusting some ‘random sliders’ which affect the output in an unpredictable manner. We discuss the act of *curation* as potentially a creative act, but not demonstrative of Meaningful Human Control over the *creation process, at an algorithmic level*. And we identify *intent, predictability, and accountability and expression* as key factors to be considered when thinking about Meaningful Human Control in a creative context.

In section 1.4: *Visual instruments: Realtime Continuous Control*, we also define what we mean by **Realtime Continuous Control** in this context, a term that we reappropriate from the *cybernetics and control theory* literature (Wiener, 1948). We further situate this in the context of *visual instruments, and realtime performative interaction*, borrowing also from visual artists developing custom performance tools ranging from Louis-Bertrand Castel (Castel, 1740), to Shuya Abe and Nam June Paik (Medienkunstnetz.de, n.d.), and the Rutt-Etra Video Synthesizer (Collopy, 2014); as well as *flow* research (Mihaly Csikszentmihalyi, 1996; Csikszentmihalyi et al., 2005) and Goal-less exploration (Secretan et al., 2008; Stanley & Lehman, 2015). We provide a more comprehensive survey of these histories in chapter 2: *Background*.

We discuss how a system that meets both of these criteria can grant opportunities for a user to explore a massive space of possibilities in realtime. Without even necessarily having an initial goal to begin with, through **Realtime Continuous Meaningful Human Control**, a user can gradually steer through the space of possibilities and hone in on some previously unknown, interesting and desirable outcomes (Secretan et al., 2008; Stanley & Lehman, 2015).

In fact, in such a system that meets these criteria, we prefer to think of the *user that interacts* with the system as more like a **performer that plays with the system**.

We believe this to be an incredibly valuable area of research. We believe this due to the decades of research in **Artistic Expressive Human-Machine Interaction** prior to DL (Krueger et al., 1985; Cadoz & Wanderley, 2000; Levin, 2000; Gillian, 2011; R. A. Fiebrink, 2011; Caramiaux, 2015). We also believe the poetically expressed words in the quote that opens this chapter, by the renowned African-American poet and writer Amiri Baraka.

7.2 Research methodology

With this thesis, we explore what DL can bring to this conversation, and we hope to encourage more researchers, artists and designers to join this conversation too.

For this reason, our aim in this thesis is not only to invent and present specific methods that bring Realtime Continuous Meaningful Human Control to Creative DL. Instead, first and foremost, our goal is to *demonstrate the potential* and wide range of possibilities granted by incorporating Realtime Continuous Meaningful Human Control into Creative DL. We wish to do this because we wish to help open up the discourse within Creative DL towards this direction. We wish to encourage more research, more work, more thinking and more conversations around these particular topics.

Based on these motivations, instead of focusing on going very deep in one direction and producing a single, very complete, thoroughly user-tested product that is ready to be deployed to the general public for widespread use, we chose to investigate many different approaches, and we develop each approach just enough to demonstrate its potential. We chose this route because we believe that this can provide a wider range of opportunities to build upon in future work.

We developed a total of five studies, and for each study we developed a software application that demonstrates a particular method. In addition to publishing peer-reviewed academic papers, we also use the software that we create, to produce a number of artworks that we share online and exhibit at galleries, museums and festivals. Furthermore, we use these artworks — and our *own experience of producing these artworks using the methods and software that we develop* — as a way of evaluating and refining our methods and software tools.

As summarised in the previous section, prior research within Creative DL did not provide *any* realtime continuous control over the outputs generated by a DNN. For this reason, the first step in evaluating the methods that we develop in this thesis can be assessed objectively, if the methods that we present do in fact introduce *any* forms of *realtime, continuous* interaction and *control* to previously *non-realtime, non-interactive, non-controllable* processes. However, this is not sufficient for *Meaningful Human Control*, and to pass the criteria that we set for ourselves in this thesis.

For this reason, through an iterative subjective evaluation process, we then fine-tune our methods and software, until we subjectively feel that they allow us to produce works that carry our *intent* and *personal expressive signature*. In other words, we create a number of artworks

with the methods and software tools that we develop in this thesis, and we use these artworks to reflect upon, test, and refine the effectiveness of our methods.

At every iteration of our software, we subjectively assess i) how ‘generic’ vs ‘unique’ the output of our software is (in other words, how easily could it have been made by other, simpler methods; or how much it resembles existing works or outputs by others); and most importantly ii) how much of our ‘intent’ is included in the output. In other words, we iteratively develop the software, continually assessing how *in control* of the output we — as an artist, and the alleged *author* of the output — feel.

It is important to note that in this context ‘being in control’ does not necessarily mean that we must have absolute control over *every aspect* of the output, and over every pixel, as this would negate the need for using such complex systems as DNNs to begin with.

As a media artist developing and working with realtime interactive computational systems for creative expression for almost two decades, there’s a number of metaphors which we have fostered over the years that adapt in context of our research within Creative DL.

The first borrows from more traditional Art practices, and in particular, Abstract Expressionism. When action painters such as Jackson Pollock or Lee Krasner splash and drip paint onto a canvas, we can think of the system that they are interacting with — comprising of a brush, paint, canvas, gravity, and fluid dynamics — as a generative system. This generative system may seem to an untrained eye (or hand) to be ‘random’, unpredictable and even *uncontrollable*. However, over time, the Artists learn to master this system, to the extent that they are able to *meaningfully control the output* such that the images that they produce carries their intent, and their personal expressive signatures.

While this metaphor may feel suitable in the context of simpler generative computational systems, once we introduce higher complexity such as DNNs, we may need suitably more complex metaphors. In this latter case, and in the context of our thesis, the desired relationship between human and machine (and DL driven software) is less analogous to the relationship between an Action Painter and their paint splashing and dripping on canvas, and it is more analogous to the relationship between a director (e.g. an Art Director or a Film Director) and a skilled crafts-person such as a graphic designer, video editor, or a cinematographer. In all of these cases, a ‘visionary’ (i.e. the director) communicates their vision to an ‘executor’ (e.g. graphic designer, video editor, or cinematographer) who produces the final output under the direction of the visionary. Needless to say, the skills and experience of the executor is critical in shaping the output. However, the influence of the executor is not only in how they produce the output. The executor may also influence the vision of the director, in that as the director sees what the executor is capable of, and how they work, their own vision may evolve.

In this sense — provided they are not micro-managing every little detail — the director is not in *full* control over every aspect of the output. However, it is their vision that is being executed by a skilled executor, and they may even continually update their vision, as they see what the executor is capable of. It’s also worth noting that the communication between director and executor is often *realtime* and *continuous*, in that the director can see the output produced by the executor while the executor is working, and the director can provide feedback, steering the results in a direction that they desire.

In the context of our research, we wish to put *human users, artists, creators* in the position of *visionary director*, and DNN powered Creative DL software tools in the position of *skilled executors*, analogous to talented graphic designers, video editors, cinematographers etc. However, we should be very clear that this analogy is merely with regards to the *relationship between director and executor*. Our goal is absolutely *not* to automate the specific tasks of graphic design, video editing or cinematography. Instead, we wish to develop these kinds of relationships between human users and semi-automated tools, while simultaneously exploring new creative roles and mediums that do not currently exist.

7.3 Summary of contributions and outcomes

As mentioned in the previous section, we developed five studies. In some ways each study builds upon the previous study, leveraging the lessons that we learnt from our experience with that study. But in many other ways, each study is quite independent and tackles the topic from very different angles.

In section 3.2: *Collaborative generative sketching with MCTS and CNNs*, our aim was to develop a collaborative generative sketching application in which a human user could collaborate with a DNN driven AI-agent in realtime. While our system showed potential, and technically worked, it did not produce the aesthetic results that we desired. To be more precise, the images produced by the agent consistently resemble noise. We discovered that this is due to the fact that *discriminative* CNN image classifiers, while performing at super-human level on *natural* images, produce many false positives on *unnatural* images. For this reason, we changed our approach, and this led to the next study that we will present. In the meantime however, the questions that we were asking in this study and the application that we were looking to build, was later replicated by researchers at Google using a very different technical approach (Ha & Eck, 2017)¹.

Despite the failure of this study to achieve the aesthetic results that we were hoping to achieve, we feel that the questions that we were asking and our proposed approach were interesting enough to be included in this thesis. And presumably for similar reasons, our paper (Akten & Grierson, 2016a) was accepted to the *Constructive Machine Learning Workshop* at the *Thirtieth Conference on Neural Information Processing Systems* (NeurIPS) in 2016. And again despite this failure, seeing our collaborative agent sketch *something* in realtime driven by a DNN with continuous human interaction and control, was exciting and encouraging enough to lay the foundations for our next study.

In section 3.3: *Realtime interactive text generation with RNN ensembles* we present a method based on an ensemble of LSTM RNNs that allows a user, or indeed a *performer*, to gesturally *conduct* the generation of text in different styles. We implement a number of different modes of interaction, ranging from playing with faders on a GUI, to gesturally and expressively conducting the generation of text through simply waving hands in the air. This latter mode of interaction, when paired with a text-to-speech system for example, removes the need for a screen altogether.

¹<https://magenta.tensorflow.org/sketch-rnn-demo>

This relieves the performer from having to sit behind a computer, and they can instead freely move around, while gesturally controlling the generation of a flow of text, just as a conductor conducts an orchestra.

It is important to note, that we do not claim this form of gestural interaction to be the most optimal mode of interaction for producing text. However, we do believe that it demonstrates perfectly what we set out to investigate in this thesis. That is, to demonstrate that it is possible to build *visual instruments* that leverage the power of generative deep models. And it is possible to design such interactive systems with both Meaningful Human Control, and Realtime Continuous Control. And that such an approach opens up a large range of new possibilities when combined with the expressive capacity of deep generative models.

We were delighted to have this work selected to be shown at the Neural Information Processing Systems (NeurIPS) 2016 conference, not just as a paper (Akten & Grierson, 2016b), but also as a live demo. In fact we showed the live demo at a number of other conferences and events. As a result, hundreds of researchers have experienced the work first-hand, and hopefully have seen the potential opportunities that *Deep Visual Instruments* have to offer. As expected, a (surprisingly small) number of researchers did not see the point of such an application. This is to be expected. We realise our research is quite far from current mainstream DL research. This demonstration in particular, gesturally conducting the generation of text, is perhaps too obscure for those researching applications of DL to Natural Language Processing (NLP) and Understanding (NLU). However, the majority of the feedback we received was incredibly positive. And seeing the trend in DL research in more recent years further confirms this.

This was the first of our studies that we consider to be fully successful in allowing Realtime Continuous Meaningful Human Control over the output of a generative system incorporating a DNN. Having accomplished this, working with relatively low-dimensional data (in the order of tens), we now switch to higher dimensions (in the order of hundreds of thousands to millions). Namely, we work with pixel based images.

In chapter 4: *Hello World: Realtime interactive training as an informative and performative tool* we develop an application that trains in realtime on a live video feed. We initially develop this application out of curiosity, in order to observe a DNN train in realtime, and hopefully gain some insights. In particular, we build our system around a Convolutional Variational Auto-Encoder (VAE). We develop an interface where we can manipulate hyperparameters in realtime via a GUI while the model is training, and we can observe the results instantly with immediate feedback. Using this system we are indeed able to observe and build a qualitative understanding of hyperparameters such as different loss functions and optimisers, varying latent dimensionality, learning-rate, momentum, gradient clipping thresholds etc.

Furthermore, an unexpected outcome of this study came as we realised the incredible performative potential of the system. We were not only able to build a qualitative understanding of these hyperparameters, but we were able to use them creatively, in an expressive and performative manner. Mapping the hyperparameters to faders on a hardware midi controller, we could manipulate multiple hyperparameters simultaneously and continuously, without taking our eyes off the screen. This allowed us to for example, find delicately balanced configurations of hyperparameters such as learning rate, momentum, and gradient clipping, such that the opti-

miser converges to *stable oscillation points*. Through further hyperparameter manipulations, we were able to cause the system to explode, and then we could bring it back under control again. Finally, on top of the hyperparameter manipulations, a second mode of interaction proved to be incredibly fruitful. This is manipulation of the video feed. In other words, simply moving ourselves or objects in front of the camera.

With these two modes of interaction combined, we felt the system really had the capacity to be *performed* and *played*, just like a *visual instrument*. In that sense this study also perfectly encapsulates our goal in this thesis. Through the bespoke system that we developed, we were able to manipulate the outputs of a generative DNN in realtime, with continuous and meaningful control. As a result, we were able to generate moving images and performances as we desired, that were not only *not possible* to create and manipulate via other means, but prior to working with this system, we did not even know that the DNN we were working with was capable of producing such images and image manipulations. This again demonstrates one of the most exciting aspects of working with systems that offer Realtime Continuous Meaningful Control.

In chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we build upon the application that we developed in the previous study, and we extend it into a new direction to allow more control over the aesthetic qualities of the images generated. We devise a system that allows us to train models on large datasets of images, that we can use in our software to transform a live video feed. We again employ the two modes of interaction that we found to be so useful in *Hello World*. The first mode of interaction involves directly manipulating the video feed which is to be transformed. This includes moving hands or objects in front of the camera, to allow for a very expressive and performative mode of interaction, resulting in a form of *digital puppetry*. Alternatively, sketching or painting on a surface that the camera is pointed to, allows for a form of *AI augmented drawing*. On top of this, we introduce a number of parameters that can be adjusted in realtime via a GUI, or faders on a hardware midi controller. This allows for more fine control over the structure and aesthetic qualities of the images generated. Combined, we found that these two modes of interaction allows for the creation of various compositions, images and videos in a very expressive and performative manner. In this respect, the system that we develop in this study again perfectly demonstrates the core principles that we investigate in this thesis.

In order to achieve this, we introduce a parametrised image processing pipeline before the DNN. This pipeline is used both during training, and during inference, and serves a number of purposes. First, during training, it acts as a form of *data augmentation*, as we randomise the parameters such that the model never sees the same inputs twice. This helps the model generalise to more novel inputs. As a result, it is more likely to perform more optimally when presented with the live video feed. Second, during inference, this pipeline transforms the live video feed into an aesthetic that is similar to the inputs that the model was trained on, further helping the model to more effectively parse the video feed in a desirable manner. Finally, during inference again, we allow for the realtime manipulation of the parameters. The effects of these parameters on the images generated are generally not very difficult to ascertain, as the filters themselves are relatively simple and deterministic. The fact that our system provides Realtime Continuous Control with instantaneous feedback, further aids in quickly building an intuition of

how the parameters influence the images generated. As a result, we found that using an image processing pipeline in this manner, and in particular, the specific image processing pipeline that we detail in the study, affords ample opportunities for Realtime Continuous Meaningful Human Control.

We develop a number of video artworks with our system and we share them online, primarily on social media. These videos have been circulated widely on both social and mainstream media. They have been exhibited in numerous galleries and exhibitions involving AI, featured in books, and even shown during Nvidia CEO Jensen Huang’s keynote at GTC (GPU Technology Conference) 2019², with the voiceover “[AI is] *inventing new ways to bring out the creative genius in us all*”. Researchers from fields ranging from philosophy to architecture to cardiovascular medicine have shown interest in these videos, asking to feature them in books and lectures. We see this as an indication that *creative human-machine collaboration* is indeed a topic that is of major interest to a broad range parties.

In creating the video artworks, the system that we developed went through many iterations. We found that even the objects that we use in front of the camera have a significant effect on the results. Particularly with respect to how much flexibility and variety they grant the user. After trying many different kinds of objects, we found that cloth is a very versatile object in this context, as they can be scrunpled up or stretched out or twisted into all kinds of different shapes. And we found that including one dark cloth and one light cloth allows the user to combine them in different ways to sculpt a variety of different scenes. Complimenting the cloth with cables, allows the user to add many different levels of flexible detail. Again we include a mixture of dark and light, thick and thin cables to allow for maximum flexibility. As an additional bit of detail, we use some cables that have larger heads on one end (e.g. ethernet cables, headphones etc.) and we cut the heads off the other end of the cable. This again provides even more flexibility for the user to sculpt different kinds of detail where needed.

As we mention in section 5.5: *Conclusion*, in the images Fig. 5.17 and Fig. 5.18 where we create bouquets of flowers from cloth and headphones, we initially had no idea that these images could be created in this manner. It was through realtime playing with our own system and trying various different objects that we had lying around on our desk, that we were noticed that the cloth-and-cables combination proved to be so versatile. When we tried placing various different objects on the cloth, we saw that the earpiece of the headphones was interpreted by our software as the central area of a flower, while the cloth was interpreted as the flower’s petals. Realising this, and further playing with the system in realtime with immediate feedback, we were able to arrange the composition of the physical objects in such a manner as to sculpt the final generated image as we desired. We could slightly crumple up the cloth and fine-tune it in this manner to appear more ‘petal-like’ to our software, while we could fine-tune the placement and orientation of the headphone cable and earpiece to maximize the overall aesthetics and ‘flower-ness’ of the final image. Again this discovery, and the images and performances that we were able to create, would not have been possible without an interaction that allowed for Realtime Continuous Meaningful Human Control.

²Nvidia GTC 2019 Keynote: <https://www.youtube.com/watch?v=Z2XlNfCtXwI&t=32>

In our final study chapter 6: *Deep Meditations: Latent storytelling*, we turn to a slightly different question. Having created a system in the previous study that allowed us to *perform* images in realtime with Meaningful Human Control, we now investigate methods of allowing users to *construct stories* in the latent space of a generative model. Specifically, we are interested in giving users the ability to freely explore this high-dimensional latent space, and then hone in on specific journeys to ultimately produce *films*. Our aim is to design a system that can provide Meaningful Human Control over the *narrative* of the films produced, as opposed to producing *random walks* in latent space, which is still the dominant paradigm when it comes to videos generated with deep generative models.

In contrast to our earlier studies where we place equal emphasis on both *Meaningful Human Control* and *Realtime Continuous Control*, for this particular study we acknowledge the role that *non-realtime, non-continuous* modes of interaction have played in some established, traditional creative workflows. We take a typical *Non-Linear Video Editing* (NLVE) workflow as base for this study, and we develop a number of stand-alone tools that integrate with an existing off-the-shelf NLVE software. This allows us to leverage the decades of research and development that went into designing and building commercial NLVE software and interfaces. As a result, we were able to develop our prototype very quickly, and we could produce a number of videos with narrative, told entirely through the latent space of generative models. We were able to synchronise these videos to music, and in fact we used our system to produce live concert visuals and a music video for the renowned electronic musician *Max Cooper* for his seminal audio-visual performance and tour *Yearning For The Infinite* which premiered at *The Barbican* in London (Cooper & Akten, 2019).

In addition, we have observed that *trajectory planning in latent space* is a very important issue if quality of both *movement* and *image* is to be preserved. This is due to the fact that the latent distribution of the generative models we are working with are typically standard normal $\mathcal{N}(0,1)$, and the mass of these distributions are concentrated in the shell of a hypersphere. For this reason, we know that using *linear interpolation* between successive latent keyframes causes the trajectory to diverge from the distribution, resulting in undesirable images. We have tested *spherical interpolation* on successive latent keyframes, and we have observed that this results in noticeable discontinuities in movement, which we find very undesirable. As a result, we have implemented two new methods, a *dynamic particle-spring based interpolation*, and a *Gaussian reprojection interpolation*. Both of these methods provide nice smooth trajectories with no noticeable discontinuities at all. However, the physical interpolation is able to produce trajectories that travel *precisely* through all desired \mathbf{z} -keyframe positions, but *not* at precisely the desired *times*. Gaussian reprojection is able to produce trajectories that travel *close* to all desired \mathbf{z} -keyframe positions, at *precisely* the desired *times*. We call this the *Heisenberg Uncertainty Principle of latent space navigation*. For this reason, we choose an interpolation method based on whether we prioritize precise *timing*, or precise *positions*. Generally we have found that the Gaussian reprojection is often sufficiently accurate in terms of position for our needs. We discuss this in more detail in subsection 6.4.8: *Trajectory planner details*.

7.4 Future directions

As we have previously mentioned, our priority in this thesis was to investigate many different approaches to Deep Visual Instruments, in order to cover as much ground as possible and maximise the likelihood of inspiring more work in these particular directions. We developed a number of different prototypes, and we chose to develop each prototype just up to the point where they could demonstrate the potential capabilities of such an approach.

For this reason, we will readily accept that the methods that we propose are just scratching the surface, and there is plenty of room to develop further. In fact, as we have mentioned numerous times, within Creative DL *Meaningful Human Control* and *Realtime Continuous Control* did not even exist at the start of our research, while today this is a rapidly growing area of inquiry. Consequently, the directions one could take from here are infinitely many. We will start by discussing some improvements that can be built upon the work that we have presented.

Of the *five* studies that we presented in this thesis, we consider *three* of them to be successful examples of *Deep Visual Instruments* which leverage the capabilities of Deep Neural Networks while providing Realtime Continuous Meaningful Human Control over the outputs generated. We discuss these in section 3.3: *Realtime interactive text generation with RNN ensembles*, chapter 4: *Hello World: Realtime interactive training as an informative and performative tool* and chapter 5: *Learning to see: Digital puppetry through realtime video transformation*. With all three of these studies, we were able to achieve our goals. Nevertheless, there are many ways in which one could build upon them. We will discuss these below.

Of the remaining two studies, section 3.2: *Collaborative generative sketching with MCTS and CNNs* did not achieve the results that we were hoping for at all, and we will discuss this shortly.

And our last study chapter 6: *Deep Meditations: Latent storytelling*, we do not consider a visual instrument, because it does not provide Realtime Continuous Control. However, for this study we intentionally wanted to replicate a typical NLVE workflow, focusing on only Meaningful Human Control. For this reason, even though this study is not an example of a visual instrument per se, we successfully achieved what we were hoping to achieve. Nevertheless, we will discuss some potential future directions for this study as well.

The first study that we presented in section 3.2: *Collaborative generative sketching with MCTS and CNNs*, did not produce the kinds of images that we were hoping for. We identified the reason for this as being due to the *discriminative* CNN that we used. We saw the potential solution for this as switching to using a *generative* LSTM RNN. Due to the lack of vector-based training data available at the time however, we switched domains, and investigated the same question operating on *text*. As a result, we produced the study that we present in section 3.3: *Realtime interactive text generation with RNN ensembles*.

Researchers at Google later demonstrated that using generative LSTM RNN models with the appropriate training data (*Quick, Draw! Dataset*, 2017; Ha & Eck, 2017), they were indeed able to fully create the functionality that we describe in this study. In particular, they use a *Sequence-to-Sequence* VAE. Such an architecture is able to model the entire space of possible sketches,

and this allows for the kinds of latent manipulations that we discuss in subsection 2.1.3: *Latent manipulations*. The researchers demonstrate this with incredible examples such as interpolating between different sketches, or performing semantic vector operations to add or remove body parts to sketches. They also demonstrate these with well developed online applications³. In this respect, we consider this particular application of collaborative generative sketching rather well explored. There can always be room for improvements of course, particularly with regards to using more state-of-the-art architectures such as a Transformer (Vaswani et al., 2017) or variants.

In section 3.3: *Realtime interactive text generation with RNN ensembles*, we use an *ensemble of LSTM RNNs*. For each *style* of text, we train a separate model. We feed the same inputs to each model, and then we perform a weighted sum of the output probability distributions. In other words, we calculate a *joint probability distribution*. We chose this approach because it allows more flexibility with regards to adding new styles to the system. For example, if we would like to add a new style after having already trained models on 20 different styles, we simply train a new model on the new dataset only. This typically takes a few hours, and then we can drop this model into our system. We do not need to re-train with all of the previous datasets as well.

However, if training time and compute resources are not a point of concern, then it can also be an option to train a single model on all of the styles collectively. In this case, one could train a *conditional* model, where each style dataset is associated with an additional input style *label*. Interpolating between styles would then be a case of interpolating between the input style vector. This could provide interesting results that differ from the method that we proposed.

Another interesting direction one could take, is to implement the variational sequence-to-sequence architecture mentioned in Sketch-RNN. This would provide a generative model whereby the entire space of possible *sentences* is captured in a static latent space. This would then allow us to perform latent manipulations such as interpolating between, or even performing semantic vector operations on sentences. This functionality has already been demonstrated with text using a very similar architecture (Kiros et al., 2015). However, our emphasis is again always on realtime performative interaction with Meaningful Human Control.

Finally, our realtime interactive system could be adapted to different domains. Perhaps one of the simplest domains which our method could be adapted to, is that of MIDI music. This would allow a performer to perform, or in this case *gesturally conduct*, music of different styles.

The interactive realtime training software that we developed in chapter 4: *Hello World: Realtime interactive training as an informative and performative tool*, allowed us to build qualitative intuitions for many of the hyperparameters involved in designing an architecture and training a model. It also allowed us to fine-tune hyperparameters in realtime such that we were able to find very particular configurations that displayed interesting behaviour, such as stable oscillations. We believe an interactive realtime training software could be beneficial to many other types of architectures, and even domains such as audio or 3D. Many modern architectures may pose a challenge with regards to performance. Even with early DCGAN architecture, our

³<https://magenta.tensorflow.org/sketch-rnn-demo>

software could not run in realtime with the hardware that was available to us. With more modern, much larger architectures, it is even less likely that performance will be adequate for realtime training. Nevertheless, we believe there is still much insight to be gained, and even performances to be performed, with perhaps ‘lighter’ versions of modern architectures adapted to run and train in realtime within such a system.

In our realtime video processing and transformation software chapter 5: *Learning to see: Digital puppetry through realtime video transformation*, we manually introduce a custom image processing pipeline before the Deep Neural Network. This grants us a number of opportunities for both Meaningful Human Control, and Realtime Continuous Control, and we summarise these in the previous section.

However, it could be questioned whether such a *manually designed* image processing pipeline needs to be introduced at all, or whether such a pipeline could be *learnt*. As we discussed in section 1.2: *Why Deep Learning?*, a *deep* Neural Network is essentially a chain of hierarchical high-dimensional transformations that are learnt end-to-end. And in fact, one of the motivations behind Deep Learning is to remove the need for such hand-crafted feature engineering pipelines. So could we remove our *custom* image processing pipeline, and instead *train* the Deep Neural Network in such a way, that the system preserves all of the functionality relating to Realtime Continuous Meaningful Human Control? We believe that at this point in time, this is not possible. However, this is exactly the area that needs more research, because we also believe that it *could* be possible. There are a number of challenges involved however.

We discuss the idea of *semantic vectors* in section 2.1.3: *Semantic latent vectors*. These are specific directions in the latent space of a generative model, such that applying these vectors to latent representations effectively apply some kind of meaningful transformation to the corresponding outputs generated by the model. For example, we can use these vectors to make an image of a face smile, or rotate an image of a car by a certain amount.

One of the key challenges of this approach, is that these semantic vectors are arbitrary directions in a very high-dimensional space, and are in no way intuitive or predictable. Outside of a few examples (Radford et al., 2015; White, 2016b), *finding* such vectors has been a relatively unexplored field. In the last few years however, interest in *semantic vector discovery* has grown rapidly (Simon, 2019; Abdal et al., 2019; Karras et al., 2019, 2020; Härkönen et al., 2020). We also believe that this is a very promising area of research.

One of the advantages of semantic vector discovery methods, is that they can be used with *pre-trained* models. In other words, semantic vector discovery can be applied to a pre-trained model that has been made available for download for example, without requiring access to the training code or methodology, or the training data. BigGAN (Brock et al., 2019) is an example of such a pre-trained model which was made available for download without access to the training code.

However, there is an issue with such semantic vector discovery methods with regards to our application. These methods are focused on *discovering* semantic latent directions. In other words, they establish key directions through analysis of the latent distribution generated by training on a particular dataset, for example via optimisation or PCA (Härkönen et al., 2020). For this reason, these methods can only discover vectors which are *already somehow present in*

the distribution of the dataset. It is by no means guaranteed that these methods will discover a vector for a particular trait that we — as a human, creative, designer — might already be visualising in our minds. This is especially true, if the particular trait that we as the human designer are imagining, does not even exist in reality (or at least in the training dataset). For example, it is very natural that a human designer might look at a picture of a nebula, and imagine that they would like to make the edges of that nebula smoother. A semantic latent vector discovery method may or may not discover such a latent direction. In fact, if the training examples do not contain images in which it is clear that some nebula have very smooth edges, while others do not, then it is almost guaranteed that a semantic vector discovery method will *not* discover such a ‘edge smoothness’ vector. In other words, such methods will not discover vectors for outputs that are *out-of-distribution*. However, as we demonstrated when discussing this study in depth in subsection 5.4.3: *Live parameter manipulation*, with our method, we can introduce filters which allow us to shape the results to construct images that do carry the desired aesthetic characteristics of the training examples, but can still be *outside* of the distribution of the training data.

One might ask, if the latent distribution of a trained model does not contain a particular desired semantic vector, could the model be re-trained in a specific way that *enforces* such desired latent vectors? The answer to this question is open. This is currently not possible at a level which can give the types of results that we demonstrate in our study. However, we do believe that this is potentially possible. Having said that, this would require a *pre-planning* of all desired parameters *before* training. In other words, before training a model, we would need to identify all of the human-understandable parameters that we desire, and only then can we train the model accordingly. If, after having trained the model, we realise that we desire a new parameter, we might need to re-train the model. Given that models can take weeks, even months to train. This is highly undesirable. With our method we do not need to retrain the model. If we desire a new human-understandable parameter, and we can envisage a solution with a simple filter, we can add it within seconds.

A final approach could be to introduce new layers to a pre-trained model. This is for example the approach taken in the very recent work by Broad et al. (2020). This method works with pre-trained models, so retraining a model to add new parameters is not necessary. The authors insert simple transformations, similar to ours, but instead of inserting the transformations *before* the trained model, they insert them as layers, in-between existing hidden layers. These transformations then directly manipulate inner representations, and can allow for out-of-distribution — but desirable — results. The authors demonstrate results with great potential, especially on simple affine transformations such as translation, rotation and scale. They also investigate applying transformations not to all of the filters within a layer, but to subsets, determined via an automated feature clustering method. This allows them to isolate and apply the transformations to only certain sections of an image. The early results of this research shows great potential, and we believe this may be a direction that can eventually produce the kinds of outputs that we demonstrate in our study.

In summary, an end-to-end deep architecture and method that can provide all of the same functionalities that we presented in our *Learning to see* study, does not yet exist. And there

are many challenges that will need to be overcome to implement such a system, but we believe this is a very interesting area of research.

For all of the studies that we have discussed so far in this section, we developed our own software from the ground up, producing a stand-alone application complete with its own GUI, and the necessary interactions and visualisations dedicated to the task at hand. For the work that we present in chapter 6: *Deep Meditations: Latent storytelling*, we take a different approach.

We chose to build our solution around an existing NLVE software. As a result, we were able to very quickly prototype a system and produce the kinds of outputs that we were looking for, as we already had an interface that provided multiple layers, powerful video editing tools, music synchronisation capabilities and many more useful features. The downside of this approach however, is that it is very difficult to seamlessly integrate new DL-specific visualisation modes that would be beneficial for such an application. For this reason, while our approach does provide *Meaningful Human Control*, the user experience is far from optimal, and involves moving back and forth between a number of separate scripts. The solution to this would be to develop a bespoke application with a dedicated custom GUI, as we have done for the other studies. This however, would undoubtedly require a far larger investment of time to develop such an application.

The highly popular and successful online app Artbreeder (Simon, 2019), is moving in this direction, and has started adding features including a very simple timeline editor to provide functionality similar to what we produced in *Deep Meditations*. At the moment, this app provides only the most basic functionality in terms of sequencing ‘keyframes’ and interpolating between them, without allowing for fine control over timing, synchronising to music, or other useful tools that established NLVE software provides. This is most probably due to the fact that building such functionality requires a much larger investment of time, as we mention above. However, we have no doubt that in the near future, either within Artbreeder, or other applications, such functionality will be implemented.

As we mentioned in the previous section, *trajectory planning in latent space* is an important topic, regardless of the UI or the host application. In addition to the methods that we have proposed, we believe a very effective method might be one that involves differential geometry and Riemannian manifolds. In this case all of the high-dimensional vector manipulations take place directly on the surface of the hypersphere. Our early tests with this method look promising, however, we do not yet have conclusive results, and we recommend this as a direction to study. Likewise another approach we believe could work, is using De Casteljaou’s algorithm with spherical interpolation. We discuss these options in more detail in section 6.5: *Conclusion*.

7.5 Final thoughts — Human-Machine Collaboration

In this thesis we have presented a number of studies that have attempted to demonstrate the potential and the possibilities that *Deep Visual Instruments* have to offer in the exploration and creation of artistic and creative works. We have designed and developed a number of realtime interactive systems that leverage the generative capabilities of DNNs trained on very large datasets. And we have tried to design these systems such that they offer **Meaningful**

Human Control, and **Realtime Continuous Control**, such that the interaction has the potential to be **expressive and performative**. Given such conditions, we think of the *user* of such as system, as more of a **performer** that **plays** the system.

With our studies, we set out to explore what is possible, and what can be achieved with DNNs within the context of **visual instruments**. We believe this is a very rich and fertile area, and we have only just scratched the surface. We do not claim to have explored each of the methods that we present to their full potential. We have not conducted user studies, and we do not make claims about optimal usability, or generalisation to a wider audience.

However, we have been able to develop and demonstrate these studies as we have intended. We have managed to introduce realtime interactivity, to previously non-realtime, non-interactive processes. Using these systems, we were able to create artworks — text, images and videos — that were previously not possible to create. We have created these outputs with Meaningful Human Control, and mostly in a Realtime Continuous manner. In other words, we consider these systems to indeed be **Deep Visual Instruments**. We have shared our results online, and they have been circulated widely, included in presentations, exhibitions, books, magazines and the subject of many conversations.

When we started our research, creative explorations within Deep Learning was very much in its infancy. And notions such as Meaningful Human Control or Realtime Continuous Control were not being discussed within creative Deep Learning research. While today, this is an active and growing area of research.

We believe that this is a very important and upcoming area of research, and we hope to have demonstrated in this thesis that such approaches can grant affordances allowing modes of media creation that were previously not possible. We believe that such approaches to Creative DL is an investigation into new mediums, new kinds of art-making, new kinds of art and performance.

This will also inevitably have a significant impact on not only individual artistic expression, but the creative industries. And as a result, we hope to see such research find its way into various kinds of media authoring software for films, games, design, music, and creative industries in general; or deployed at mass scale for mainstream use through online or mobile applications and ‘smart filters’ on social media. And we especially hope that the work that we have presented in this thesis can serve as a starting point and inspiration for similar future work and contribute to this rapidly blossoming field.

We believe this topic of **Human-Machine Collaboration** is a very crucial one. Not only in the production of artistic and creative works, as is the topic of this thesis, but beyond that. As AI enters every aspect of our lives, **Meaningful Human Control** and **Realtime Continuous Control**, are valuable design principles that we believe should not be overlooked.

References

- Abadi, M., & Others, A. (2015). *TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Retrieved from <http://tensorflow.org>
- Abdal, R., Qin, Y., & Wonka, P. (2019). Image2StyleGAN: How to embed images into the StyleGAN latent space? *Proceedings of the IEEE International Conference on Computer Vision, 2019-October*, 4431–4440. doi: 10.1109/ICCV.2019.00453
- Akten, M. (2015). ofxMSAmcts. Retrieved from <https://github.com/memo/ofxmsamcts>
- Akten, M. (2016). *ofxMSATensorFlow*. Retrieved from <https://github.com/memo/ofxMSATensorFlow>
- Akten, M. (2017). *webcam-pix2pix-tensorflow*. Retrieved from <https://github.com/memo/webcam-pix2pix-tensorflow>
- Akten, M. (2018). *py-msa-kdenlive*. Retrieved from <https://github.com/memo/py-msa-kdenlive>
- Akten, M., & Cooper, M. (2020). "Morphosis" Music video. Retrieved from <https://www.youtube.com/watch?v=7oWjgbCXp-o>
- Akten, M., Fiebrink, R., & Grierson, M. (2018). Deep Meditations: Controlled navigation of latent space. *Machine Learning for Creativity and Design - NIPS 2018 Workshop*.
- Akten, M., Fiebrink, R., & Grierson, M. (2019). Learning to see: You are what you see. *ACM SIGGRAPH 2019 Art Gallery, SIGGRAPH 2019*, 1–6. doi: 10.1145/3306211.3320143
- Akten, M., & Grierson, M. (2016a). Collaborative creativity with Monte-Carlo Tree Search and Convolutional Neural Networks. In *Nips 2016, constructive machine learning workshop*.
- Akten, M., & Grierson, M. (2016b). Real-time interactive sequence generation and control with Recurrent Neural Network ensembles. *NIPS 2016, Recurrent Neural Networks Symposium, Poster and demo presentation(Nips)*. Retrieved from <http://arxiv.org/abs/1612.04687>
- Al-rifaie, M. M., & Bishop, J. M. (2015). Weak and Strong Computational Creativity. In *Computational creativity research: Towards creative machines* (Vol. 7, pp. 0–14).
- Alves, B. (2005). Digital Harmony of Sound and Light. *Computer Music Journal*, 29(4), 45–54.
- Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., & de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*.
- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. Retrieved from <http://arxiv.org/abs/1701.07875>
- Bailey, J. (2020). The Tools of Generative Art, from Flash to Neural Networks. *Art in America*(January), 34–41. Retrieved from <https://www.artnews.com/>

- art-in-america/features/generative-art-tools-flash-processing-neural-networks-1202674657/
- Baraka, I. A. (1969). Technology & Ethos. Vol. 2 Book of Life. In *Raise rage rays raze: Essays since 1965*.
- Barry, S., & Kim, Y. (2018). Style Transfer for Musical Audio Using Multiple Time-Frequency Representations. *OpenReview:BybQ7zWCb*, *V*(1), 1–11.
- Bau, D., Strobelt, H., Peebles, W., Wulff, J., Zhou, B., Zhu, J. Y., & Torralba, A. (2019). Semantic photo manipulation with a generative image prior. *ACM Transactions on Graphics*, *38*(4). doi: 10.1145/3306346.3323023
- Bellman, R. (1957). *A Markovian decision process* (Tech. Rep.). DTIC Document.
- Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *Tpami*(1993), 1–30.
- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.
- Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*, 281–305.
- Berthelot, D., Schumm, T., & Metz, L. (2017). BEGAN: Boundary Equilibrium Generative Adversarial Networks. , 1–10. Retrieved from <http://arxiv.org/abs/1703.10717> doi: 1703.10717
- Bevilacqua, F., & Muller, R. (2005). A gesture follower for performing arts. *Proceedings of the International Gesture . . .*, 3–4.
- Bevilacqua, F., Zamborlin, B., Sypniewski, A., Schnell, N., Guédy, F., & Rasamimanana, N. (2009). Continuous realtime gesture following and recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5934 LNAI*, 73–84.
- Bishop, C. M. (1994). Mixture density networks.
- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Boden, M. A. (1998). 18 Computer Models of Creativity. *Handbook of creativity*, 351.
- Boden, M. A. (2004). *The creative mind: Myths and mechanisms*. Psychology Press. doi: 10.4324/9780203508527
- Boulanger-Lewandowski, N., Vincent, P., & Bengio, Y. (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *arXiv preprint arXiv:1206.6392*.
- Broad, T., Leymarie, F. F., & Grierson, M. (2020). Network Bending: Manipulating The Inner Representations of Deep Generative Models. *arXiv preprint arXiv:2005.12420*.
- Brock, A., Donahue, J., & Simonyan, K. (2019). Large scale GaN training for high fidelity natural image synthesis. *7th International Conference on Learning Representations, ICLR 2019*, 1–35.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. Retrieved from <http://arxiv.org/abs/2005.14165>
- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., . . .

- Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1–43.
- Cadoz, C., & Wanderley, M. (2000). Gesture-music. *Trends in gestural control of music*, 71–94. Retrieved from [http://www.vigliensoni.com/McGill/CURSOS/2009_09/MUMT620/READINGS/2/2_Gesture-Music\(Cadoz-Wanderley\).pdf](http://www.vigliensoni.com/McGill/CURSOS/2009_09/MUMT620/READINGS/2/2_Gesture-Music(Cadoz-Wanderley).pdf)
- Camurri, A., Mazzarino, B., Ricchetti, M., Timmers, R., & Volpe, G. (2004). Multimodal analysis of expressive gesture in music and dance performances. In *Gesture-based Communication in Human-Computer Interaction, LNAI 2915* (pp. 20–39). doi: 10.1007/978-3-540-24598-8_3
- Candy, L. (2006). Practice Based Research: A Guide. *CCS report*, 1(2).
- Caplan, L. (2020). The Social Conscience of Generative Art. *Art in America*(January), 50–57. Retrieved from <https://www.artnews.com/art-in-america/features/max-bense-gustav-metzger-generative-art-1202674265/>
- Caramiaux, B. (2015). Motion Modeling for Expressive Interaction A Design Proposal using Bayesian Adaptive Systems. In *International workshop on movement and computing (moco)* (Vol. 5). IRCAM.
- Caramiaux, B., Montecchio, N., Tanaka, A., & Bevilacqua, F. (2014). Adaptive Gesture Recognition with Variation Estimation for Interactive Systems. *ACM Transactions on Interactive Intelligent Systems (TiiS) (In Press)*, V(212). doi: 10.1145/2643204
- Caramiaux, B., & Tanaka, A. (2013). Machine Learning of Musical Gestures. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 513–518. Retrieved from <http://nime2013.kaist.ac.kr/>
- Cassell, J., & McNeill, D. (1991). Gesture and the Poetics of Prose. *Poetics Today*, 12(3), 375–404. doi: 10.2307/1772644
- Castel, L.-B. (1740). *L’Optique des Couleurs*. Paris: Chez Briasson.
- Cavallo, F., Pease, A., Gow, J., & Colton, S. (2013). Using Theory Formation Techniques for the Invention of Fictional Concepts. , 176–183.
- Champanard, A. J. (2016). Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks. *arXiv preprint arXiv:1603.01768*.
- Chen, X., Duan, Y., Houthoofd, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. Retrieved from <http://arxiv.org/abs/1606.03657>
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1724–1734). Association for Computational Linguistics.
- Cohen, H. (1973). Parallel to perception: some notes on the problem of machine-generated art. *Computer Studies*, 1–10. Retrieved from <http://haroldcohen.com/aaron/publications/paralleltoperception.pdf>
- Cohen, H. (1994). The Further Exploits of Aaron, Painter.
- Cohen, H. (2006). *AARON, Colorist: from Expert System to Expert*.

- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 1(12), 2493–2537. Retrieved from <http://dl.acm.org/citation.cfm?id=2078186>
- Collopy, P. S. (2014). Video synthesizers: From analog computing to digital art. *IEEE Annals of the History of Computing*, 36(4), 74–86. doi: 10.1109/MAHC.2014.62
- Colton, S., Goodwin, J., & Veale, T. (2012). Full-FACE Poetry Generation. *Proceedings of the Third International Conference on Computational Creativity (ICCC'12)*, 95–102.
- Colton, S., Pease, A., & Charnley, J. (2011). Computational creativity theory: The FACE and IDEA descriptive models. *Proceedings of the Second International Conference on Computational Creativity*, 90–95. Retrieved from [https://www.doc.ic.ac.uk/~\sim\\$jwc04/papers/conferences/colton.iccc11.pdf](https://www.doc.ic.ac.uk/~\sim$jwc04/papers/conferences/colton.iccc11.pdf)
- Colton, S., & Wiggins, G. a. (2012). Computational creativity: The final frontier? *Frontiers in Artificial Intelligence and Applications*, 242, 21–26. doi: 10.3233/978-1-61499-098-7-21
- Cook, J. D. (2011). *Willie Sutton and the multivariate normal distribution*. Retrieved from <https://www.johndcook.com/blog/2011/09/01/multivariate-normal-shell/>
- Cook, M., Colton, S., & Gow, J. (2014). Automating Game Design In Three Dimensions. *AISB Symposium on AI and Games*, 3–6.
- Cooper, M., & Akten, M. (2019). "Yearning For The Infinite" Audio-Visual performance. Retrieved from <https://www.yearningfortheinfinite.net/>
- Cope, D. H. (2010). *Recombinant music composition algorithm and method of using the same*.
- Corneli, J., Jordanous, A., Guckelsberger, C., Pease, A., & Colton, S. (2014). Modelling serendipity in a computational context. *arXiv preprint arXiv:1411.0440*.
- Coupric, C., Najman, L., & Lecun, Y. (2013). Learning Hierarchical Features for Scene Labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8), 1915–1929. doi: 10.1109/TPAMI.2012.231
- Crnkovic-friis, L., & Crnkovic-friis, L. (2016). Generative Choreography using Deep Learning. *arXiv preprint arXiv:1605.06921*.
- Csikszentmihalyi, M., Nakamura, J., & Abuhamdeh, S. (2005). Flow. In *Handbook of competence and motivation* (pp. 598–608). New York: Harper & Row. doi: 10.1007/978-94-017-9088
- Cui, B., Qi, C., & Wang, A. (2017). Multi-style Transfer: Generalizing Fast Style Transfer to Several Genres. , 2017. Retrieved from <http://cs231n.stanford.edu/reports/2017/pdfs/401.pdf> doi: 10.1016/j.ecoenv.2016.05.022
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Ieee conference on computer vision and pattern recognition, 2009. cvpr 2009*. (pp. 248–255). IEEE.
- Deng, L., Hinton, G., & Kingsbury, B. (2013). New Types of Deep Neural Network Learning for Speech Recognition and Related Applications : an Overview. , 8599–8603.
- Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: A Generative Model for Music. Retrieved from <https://github.com/openai/jukebox>.
- Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using Real NVP. Retrieved from <http://arxiv.org/abs/1605.08803> doi: 1605.08803
- Dong, J., Gigan, S., Krzakala, F., & Wainrib, G. (2016). Scaling up Echo-State Networks with

- multiple light scattering. Retrieved from <http://arxiv.org/abs/1609.05204>
- Dosovitskiy, A., & Brox, T. (2015). Inverting Convolutional Networks with Convolutional Networks. , 1–15.
- Dourish, P. (2001). *Where the Action Is: The Foundations of Embodied Interaction* (Vol. 36) (No. 3). Retrieved from <http://books.google.com/books?id=DCIy2zxrCqcC&pgis=1> doi: 10.1162/leon.2003.36.5.412
- Draves, S. (2005). The Electric Sheep screen-saver: A case study in aesthetic evolution. *Proc. EvoMUSART*, 458–467. Retrieved from http://dx.doi.org/10.1007/978-3-540-32003-6_46 doi: 10.1007/978-3-540-32003-6_46
- Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., & Courville, A. (2017). Adversarially learned inference. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.
- Eck, D., & Schmidhuber, J. (2002). A First Look at Music Composition using LSTM Recurrent Neural Networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103.
- Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., & Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. In *Iclr 2019*.
- Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network. *Bernoulli*(1341), 1–13. Retrieved from <http://igva2012.wikispaces.asu.edu/file/view/Erhan+2009+Visualizing+higher+layer+features+of+a+deep+network.pdf>
- Fails, J. A., Olsen, Jr., D. R., & Olsen, D. R. (2003). Interactive Machine Learning. In *Proceedings of the 8th international conference on intelligent user interfaces* (pp. 39–45). Miami, Florida, USA: ACM. Retrieved from <http://portal.acm.org/citation.cfm?doid=604045.604056> doi: 10.1145/604045.604056
- Fels, S. S., & Hinton, G. E. (1993). Glove-talk: a neural network interface between a data-glove and a speech synthesizer. *IEEE Transactions on Neural Networks*, 4(1), 2–8. doi: 10.1109/72.182690
- Fiebrink, R., Trueman, D., & Cook, P. (2009). A metainstrument for interactive, on-the-fly machine learning. *Proc. NIME*, 2, 3. Retrieved from [http://www.cs.dartmouth.edu/\\$\sim\\$cs104/BodyPartRecognition.pdf%5Cnhttp://www.cs.princeton.edu/\\$\sim\\$fiebrink/publications/FiebrinkTruemanCook_NIME2009.pdf](http://www.cs.dartmouth.edu/\simcs104/BodyPartRecognition.pdf%5Cnhttp://www.cs.princeton.edu/\simfiebrink/publications/FiebrinkTruemanCook_NIME2009.pdf)
- Fiebrink, R. A. (2011). Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance. *Imagine*(January), 376. Retrieved from [http://www.cs.princeton.edu/\\$\sim\\$fiebrink/drop/thesis/thesis.kbow_conclusions.pdf](http://www.cs.princeton.edu/\simfiebrink/drop/thesis/thesis.kbow_conclusions.pdf)
- Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *34th International Conference on Machine Learning, ICML 2017*, 3, 1856–1868.
- Fuegi, J., & Francis, J. (2003). Lovelace & Babbage and the creation of the 1843 'Notes'. *IEEE Annals of the History of Computing*, 25(4), 16–26. Retrieved from <https://www.scss.tcd.ie/coghlan/repository/J.Byrne/A.Lovelace/J.Fuegi.&J.Francis.2003.pdf>
- Gatys, L. A., Ecker, A. S., & Bethge, M. (2015a). A Neural Algorithm of Artistic Style. *arXiv preprint arXiv:1508.06576*.

- Gatys, L. A., Ecker, A. S., & Bethge, M. (2015b). Texture Synthesis Using Convolutional Neural Networks. *Nips*, 1–10. Retrieved from <http://arxiv.org/abs/1505.07376>
- Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural networks, 2000. ijcnn 2000, proceedings of the ieee-inns-enns international joint conference on* (pp. 189–194). IEEE.
- Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10), 2451–2471.
- Gillian, N. E. (2011). Gesture Recognition for Musician Computer Interaction. *Social Sciences*(March).
- Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, 9, 249–256.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from <http://www.deeplearningbook.org>
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 2672–2680. Retrieved from <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
- Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. , 1–11. Retrieved from <http://arxiv.org/abs/1412.6572>
- Graves, A. (2008). *Supervised Sequence Labelling with Recurrent Neural Networks* (Unpublished doctoral dissertation).
- Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.
- Graves, A. (2013). Generating sequences with Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*.
- Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), 855–868.
- Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 1–26. Retrieved from <http://arxiv.org/abs/1410.5401>
- Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *arXiv preprint arXiv:1503.04069*, 10. Retrieved from <http://arxiv.org/abs/1503.04069> doi: 10.1017/CBO9781107415324.004
- Gregor, K., Danihelka, I., Graves, A., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *arXiv preprint arXiv:1502.04623*.
- Grierson, M. (2005). *Audiovisual composition* (Doctoral dissertation). Retrieved from <http://www.strangeloop.co.uk/Dr.M.Grierson-AudiovisualCompositionThesis.pdf>
- Guzella, T. S., & Caminhas, W. M. (2009). A review of machine learning approaches to Spam filtering. *Expert Systems with Applications*, 36(7), 10206–10222. Retrieved from <http://dx.doi.org/10.1016/j.eswa.2009.02.037> doi: 10.1016/j.eswa.2009.02.037
- Gwak, J., Choy, C. B., Garg, A., Chandraker, M., & Savarese, S. (2017). Weakly Supervised Generative Adversarial Networks for 3D Reconstruction. Retrieved from <http://arxiv>

- .org/abs/1705.10904
- Ha, D. (2015). *Recurrent Net Dreams Up Fake Chinese Characters in Vector Format with TensorFlow*. Retrieved from <http://blog.otoro.net/2015/12/28/recurrent-net-dreams-up-fake-chinese-characters-in-vector-format-with-tensorflow/>
- Ha, D., & Eck, D. (2017). A Neural Representation of Sketch Drawings. , 1–20. Retrieved from <http://arxiv.org/abs/1704.03477>
- Härkönen, E., Hertzmann, A., Lehtinen, J., & Paris, S. (2020). GANSpace: Discovering Interpretable GAN Controls. , 1–14. Retrieved from <http://arxiv.org/abs/2004.02546>
- Hertzmann, A. (2001). *Algorithms for Rendering in Artistic Styles* (Doctoral dissertation). doi: <http://doi.acm.org/10.1145/933267>
- Hesse, C. (2017). *edges2cats Image-to-Image demo*. Retrieved from <https://affinelayer.com/pixsrv/>
- Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., ... Lerchner, A. (2017). B-VAE: Learning basic visual concepts with a constrained variational framework. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*(July), 1–13. Retrieved from <https://openreview.net/forum?id=Sy2fzU9g1>
- Higham, T., Basell, L., Jacobi, R., Wood, R., Ramsey, C. B., & Conard, N. J. (2012). Testing models for the beginnings of the Aurignacian and the advent of figurative art and music: The radiocarbon chronology of Geißenklösterle. *Journal of Human Evolution*, 62(6), 664–676. doi: 10.1016/j.jhevol.2012.03.003
- Hindupur, A. (n.d.). *The GAN Zoo*. Retrieved 2020-08-01, from <https://github.com/hindupuravinash/the-gan-zoo>
- Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., ... Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 29(6), 82–97.
- Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen* (Unpublished doctoral dissertation). Technische Universität München.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
- Hornik, K. (1991). Approximation Capabilities of Multilayer Neural Network. *Neural Networks*, 4(1991), 251–257.
- Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., ... Ng, A. Y. (2015). An Empirical Evaluation of Deep Learning on Highway Driving. , 1–7. Retrieved from <http://arxiv.org/abs/1504.01716>
- Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *arXiv preprint arXiv:1611.07004*.
- Jiang, L., Zhang, C., Huang, M., Liu, C., Shi, J., & Loy, C. C. (2020). TSIT: A Simple and Versatile Framework for Image-to-Image Translation. Retrieved from <http://arxiv.org/abs/2007.12072>
- Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. *European conference on computer vision*, 694–711.
- Jordanous, A. (2014). *What is Computational Creativity?* Retrieved from <http://>

- www.creativitypost.com/science/what_is_computational_creativity
- Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). *An Empirical Exploration of Recurrent Network Architectures* (Vol. 37).
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning : A Survey. *Journal of artificial intelligence research*, 4, 237–285. Retrieved from <http://arxiv.org/abs/cs/9605103>
- Karpathy, A. (2015a). *char-rnn*. Retrieved from <https://github.com/karpathy/char-rnn>
- Karpathy, A. (2015b). *The Unreasonable Effectiveness of Recurrent Neural Networks*. Retrieved from <http://karpathy.github.io/2015/05/21/rnn-effectiveness>
- Karpathy, A., Johnson, J., & Li, F.-F. (2015). Visualizing and Understanding Recurrent Networks. *arXiv preprint arXiv:1506.02078*.
- Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv preprint arXiv:1710.10196*. Retrieved from <http://arxiv.org/abs/1710.10196>
- Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019-June*, 4396–4405. doi: 10.1109/CVPR.2019.00453
- Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and Improving the Image Quality of StyleGAN. Retrieved from <http://arxiv.org/abs/1912.04958>
- Katan, S., Grierson, M., & Fiebrink, R. (2015). Using Interactive Machine Learning to Support Interface Development Through Workshops with Disabled People. In *Chi '15 proceedings of the 33rd annual acm conference on human factors in computing systems*.
- Kiefer, C. (2014). Musical Instrument Mapping Design with Echo State Networks. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 293–298. Retrieved from http://www.nime.org/proceedings/2014/nime2014_530.pdf
- Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1–13. Retrieved from <http://arxiv.org/abs/1412.6980>
- Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1x1 Convolutions. , 1–15. Retrieved from <http://arxiv.org/abs/1807.03039> doi: 10.1097/PHM.0b013e318038d39c
- Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*. Retrieved from <http://arxiv.org/abs/1312.6114>
- Kiros, R. (2015). *neural-storyteller*. Retrieved from <https://github.com/ryankiros/neural-storyteller>
- Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., & Fidler, S. (2015). Skip-thought vectors. *Advances in Neural Information Processing Systems, 2015-Janua*(786), 3294–3302.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, 1–9.
- Krueger, M. W., Gionfriddo, T., & Hinrichsen, K. (1985). VIDEOPLACE—an artificial reality.

- ACM SIGCHI Bulletin*, 16(4), 35–40. doi: 10.1145/1165385.317463
- Kyprianidis, J. E., Collomosse, J., Wang, T., & Isenberg, T. (2013). State of the 'Art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5), 866–885. doi: 10.1109/TVCG.2012.160
- Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. *33rd International Conference on Machine Learning, ICML 2016*, 4, 2341–2349. Retrieved from <http://arxiv.org/abs/1512.09300>
- Laurens van der Maaten, & Geoffrey Hinton. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605.
- LaViola Jr., J. J. (2014). *An Introduction to 3D Gestural Interfaces*. Retrieved from <http://doi.acm.org/10.1145/2614028.2615424> doi: 10.1145/2614028.2615424
- LeapMotion. (n.d.). *LeapMotion*. Retrieved from <https://www.leapmotion.com/>
- LeCun, Y. (2012). Learning Invariant Feature Hierarchies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7583 LNCS(PART 1), 496–505. doi: 10.1007/978-3-642-33863-2_51
- LeCun, Y. (2014). The Unreasonable Effectiveness of Deep Learning. In *Facebook ai research & center for data science, nyu*.
- LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 255–258. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.9297&rep=rep1&type=pdf>
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 541–551.
- LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database. *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>.
- Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., ... Shi Twitter, W. (2017). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *Cvpr*, 4681–4690. Retrieved from http://openaccess.thecvf.com/content_cvpr_2017/papers/Ledig_Photo-Realistic_Single_Image_CVPR.2017_paper.pdf<http://arxiv.org/abs/1609.04802> doi: 10.1109/CVPR.2017.19
- Lee, M., Freed, A., & Wessel, D. (1992). Neural networks for simultaneous classification and parameter estimation in musical instrument control. *Proceedings of SPIE*, 1706, 244–255. Retrieved from <http://link.aip.org/link/?PSI/1706/244/1&Agg=doi> doi: 10.1117/12.139949
- Levin, G. (2000). Painterly Interfaces for Audiovisual Performance. *Media*, 1–151. Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Painterly+Interfaces+for+Audiovisual+Performance#0>
- Liang, Y., Yang, Z., Zhang, K., He, Y., Wang, J., & Zheng, N. (2017). Single Image Super-resolution via a Lightweight Residual Convolutional Neural Network. , 1–11. Retrieved from <http://arxiv.org/abs/1703.08173>
- Lieberman, Z., Watson, T., & Castro, A. (2016). *OpenFrameworks*. Retrieved from [www](http://www.openframeworks.cc/)

- .openframeworks.cc
- Lim, B., Son, S., Kim, H., Nah, S., & Lee, K. M. (2017). Enhanced Deep Residual Networks for Single Image Super-Resolution. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2017-July*, 1132–1140. doi: 10.1109/CVPRW.2017.151
- Lin, C., Liu, D., & Kelly, A. (2016). Deep Adversarial 3D Shape Net. *Report CMU(2013)*, 1–9.
- Lin, H. W., & Tegmark, M. (2016). Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225*.
- Lipton, Z. C., & Tripathi, S. (2017). Precise Recovery of Latent Vectors from Generative Adversarial Networks. *arXiv preprint arXiv:1702.04782*. Retrieved from <http://arxiv.org/abs/1702.04782>
- Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., . . . Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42(1995), 60–88. doi: 10.1016/j.media.2017.07.005
- Liu, J., Yu, F., & Funkhouser, T. (2017). Interactive 3D Modeling with a Generative Adversarial Network. Retrieved from <http://arxiv.org/abs/1706.05170>
- Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width. *Advances in Neural Information Processing Systems, 2017-Decem(Nips)*, 6232–6240.
- Mahendran, A., & Vedaldi, A. (2014). Understanding Deep Image Representations by Inverting Them.
- Mccormack, J. (2014). Balancing Act : Variation and Utility in Evolutionary Art. , 26–37.
- McCormack, J., Bown, O., Dorin, A., McCabe, J., Monro, G., & Whitelaw, M. (2014). Ten Questions Concerning Generative Computer Art. *Leonardo*, 47(2), 135–141. Retrieved from http://www.mitpressjournals.org/doi/abs/10.1162/LEON_a_00533 doi: 10.1162/LEON_a_00533
- McCormack, J., & D’Inverno, M. (2012). Computers and Creativity: The Road Ahead. *Computers and Creativity*, 421–424. Retrieved from <http://dx.doi.org/10.1007/978-3-642-31727-9> doi: 10.1007/978-3-642-31727-9
- McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Idea Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5, 115–133. Retrieved from [http://www.cse.chalmers.se/~\sim\\$coquand/AUTOMATA/mcp.pdf](http://www.cse.chalmers.se/~\sim$coquand/AUTOMATA/mcp.pdf) doi: 10.1007/BF02478259
- McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv*.
- McNeill, D., & Levy, E. (1980). *Conceptual representations in language activity and gesture* (Tech. Rep.). Spencer Foundation, Chicago, IL.; National Inst. of Mental Health (DHHS), Bethesda, MD.
- Medienkunstnetz.de. (n.d.). *Paik/Abe Synthesizer*. Retrieved 2020-04-03, from <http://medienkunstnetz.de/works/paik-abe-synthesizer/>
- Mihaly Csikszentmihalyi. (1996). *Creativity: Flow and the psychology of discovery and invention*.

- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, 1–9. doi: 10.1162/jmlr.2003.3.4-5.951
- Miolane, N., Mathe, J., Donnat, C., Jorda, M., & Penneç, X. (2018). geomstats: a Python Package for Riemannian Geometry in Machine Learning. Retrieved from <http://arxiv.org/abs/1805.08308>
- Mital, P. K. (2017). Time Domain Neural Audio Style Transfer. *Machine Learning for Creativity and Design - NIPS 2017 Workshop*(Nips). Retrieved from <http://arxiv.org/abs/1711.11160>
- Mitchell, T. M. (1997). *Machine Learning* (Vol. 1) (No. 3). McGraw Hill. Retrieved from <http://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/0070428077> doi: 10.1007/BF00116892
- Mitra, S., & Acharya, T. (2007). Gesture Recognition : A Survey. *IEEE Transactions On Systems, Man, And Cybernetics - Part C: Applications And Reviews*, 37(3), 311–324. doi: 10.1109/TSMCC.2007.893280
- Mordvintsev, A., Olah, C., & Tyka, M. (2015). *Deepdream*. Retrieved from <http://googleresearch.blogspot.ch/2015/06/inceptionism-going-deeper-into-neural.html>
- Mordvintsev, A., Pezzotti, N., Schubert, L., & Olah, C. (2018). Differentiable Image Parameterizations. *Distill*, 3(7). doi: 10.23915/distill.00012
- Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 807–814).
- Nayebi, A., & Vitelli, M. (2015). GRUV : Algorithmic Music Generation using Recurrent Neural Networks. Retrieved from <https://cs224d.stanford.edu/reports/NayebiAran.pdf>
- Ng, A. (2013). Machine Learning and AI via Brain Simulations. In *Stanford university*. Retrieved from <http://www.youtube.com/watch?v=n1ViNeWhC24%5Cnpapers3://publication/uuid/FD4F0FE0-BB23-431D-8840-6808135CC089>
- Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (Vol. 07-12-June, pp. 427–436). doi: 10.1109/CVPR.2015.7298640
- Odena, A., Olah, C., & Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. *34th International Conference on Machine Learning, ICML 2017*, 6, 4043–4055.
- ofxAddons. (2015). Retrieved from <http://ofxaddons.com/>
- Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature Visualization. *Distill*, 2(11). doi: 10.23915/distill.00007
- Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., & Mordvintsev, A. (2018). The Building Blocks of Interpretability. *Distill*, 3(3). doi: 10.23915/distill.00010
- Pachet, F. (2003). The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3), 333–341. doi: 10.1076/jnmr.32.3.333.16861
- Papert, S., Valente, J. A., & Bitelman, B. (1980). *Logo: computadores e educa*. Brasiliense.
- Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with

- spatially-adaptive normalization. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019-June*, 2332–2341. doi: 10.1109/CVPR.2019.00244
- Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. *Computing Research Repository (CoRR) abs/1211.5063*.
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning icml* (Vol. 28, pp. 1310–1318).
- Penny, S. (2017). *Making sense: Cognition, computing, art, and embodiment*. MIT Press.
- Perlin, K. (2002). Improving Noise. *ACM Transactions on Graphics*, 681–682. doi: 10.1145/566570.566636
- Pham, V., Bluche, T., Kermorvant, C., & Louradour, J. (2014). Dropout Improves Recurrent Neural Networks for Handwriting Recognition. In *Frontiers in handwriting recognition (icfhr), 2014 14th international conference on* (pp. 285–290). IEEE.
- Pike, A. W., Hoffmann, D. L., García-Diez, M., Pettitt, P. B., Alcolea, J., De Balbín, R., ... Zilhão, J. (2012). U-series dating of paleolithic art in 11 caves in Spain. *Science*, 336(6087), 1409–1413. doi: 10.1126/science.1219957
- Plummer-Fernandez, M. (2020). Bots vs. AI: Two Kinds of Software Art Take Different Approaches to the Digital Commons. *Art in America*(January), 58–63. Retrieved from <https://www.artnews.com/art-in-america/features/bots-ai-art-digital-commons-1202675976/>
- Quick, Draw! Dataset*. (2017). Retrieved from <https://github.com/googlecreativelab/quickdraw-dataset>
- Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.
- Rezende, D. J., Mohamed, S., Danihelka, I., Gregor, K., & Wierstra, D. (2016). One-Shot Generalization in Deep Generative Models.
- Rezende, D. J., & Viola, F. (2018). Taming VAEs. Retrieved from <http://arxiv.org/abs/1810.00597>
- Ridgeway, K. (2016). A Survey of Inductive Biases for Factorial Representation-Learning. , 1–36. Retrieved from <http://arxiv.org/abs/1612.05299>
- Risser, E., Wilmot, P., & Barnes, C. (2017). Stable and Controllable Neural Texture Synthesis and Style Transfer Using Histogram Losses. *arXiv preprint arXiv:1701.08893*.
- Roff, H. M., & Moyes, R. (2016). *Meaningful Human Control , Artificial Intelligence and Autonomous Weapons* (Tech. Rep.).
- Rokeby, D. (1986). *Very Nervous System*.
- Ruder, M., Dosovitskiy, A., & Brox, T. (2016). Artistic style transfer for videos. *German Conference on Pattern Recognition*.
- Ruder, M., Dosovitskiy, A., & Brox, T. (2018). Artistic style transfer for videos and spherical images. *International Journal of Computer Vision*, 126(11), 1199–1219.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning Internal Representations by Error Propagation* (Tech. Rep.).
- Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., & Hadsell, R.

- (2019). Meta-learning with latent embedding optimization. *7th International Conference on Learning Representations, ICLR 2019*, 1–17.
- Sahoo, D., Pham, Q., Lu, J., & Hoi, S. C. (2018). Online deep learning: Learning deep neural networks on the fly. *IJCAI International Joint Conference on Artificial Intelligence, 2018-July*, 2660–2666. doi: 10.24963/ijcai.2018/369
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved Techniques for Training GANs. , 1–10. Retrieved from <http://arxiv.org/abs/1606.03498> doi: arXiv:1504.01391
- Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). One-shot Learning with Memory-Augmented Neural Networks. Retrieved from <http://arxiv.org/abs/1605.06065>
- Scharre, P., & Horowitz, M. C. (2015). *Meaningful human control in weapon systems: A primer*.
- Schedel, M., Fiebrink, R., & Perry, P. (2011). Wekinating 000000Swan : Using Machine Learning to Create and Control Complex Artistic Systems. *Proceedings of the International Conference on New Interfaces for Musical Expression*(June), 453–456. Retrieved from <http://www.nime2011.org/proceedings/papers/M10-Schedel.pdf>
- Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2), 234–242.
- Schmidhuber, J. (2006). Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2), 173–187. doi: 10.1080/09540090600768658
- Schmidhuber, J. (2007). Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity and creativity. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4754 LNAI(September), 32–33. Retrieved from <http://arxiv.org/abs/0709.0674> doi: 10.1007/978-3-540-75225-7_6
- Schmidhuber, J. (2009a). Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5499 LNAI(April 2009), 48–76. doi: 10.1007/978-3-642-02565-5_4
- Schmidhuber, J. (2009b). Simple Algorithmic Theory of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes. *Journal of SICE*, 48, 21–32. doi: http://dx.doi.org/10.1007/978-3-642-02565-5_4
- Schmidhuber, J. (2010a). Artificial scientists & artists based on the formal theory of creativity. *Artificial General Intelligence - Proceedings of the Third Conference on Artificial General Intelligence, AGI 2010*, 145–150. Retrieved from http://agi-conf.org/2010/wp-content/uploads/2009/06/paper_46.pdfhttp://agi-conf.org/2010/wp-content/uploads/2009/06/paper%7B_%7D46.pdf doi: 10.2991/agi.2010.32
- Schmidhuber, J. (2010b). Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2(3), 230–247. doi: 10.1109/TAMD.2010.2056368

- Schmidhuber, J. (2012a). A formal theory of creativity to model the creation of art. *Computers and Creativity*, 9783642317, 323–337. doi: 10.1007/978-3-642-31727-9_12
- Schmidhuber, J. (2012b). Self-Delimiting Neural Networks. *arXiv preprint arXiv:1210.0118*.
- Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61, 85–117. doi: 10.1016/j.neunet.2014.09.003
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 2673–2681. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=650093
- Searle, J. R. (1980). Minds, Brains, and Programs. *Behavioral and Brain Sciences*, 3, 1–19. doi: 10.1017/S0140525X00005756
- Secretan, J., Beato, N., D’Ambrosio, D. B., Rodriguez, A., Campbell, A., & Stanley, K. O. (2008). Picbreeder: Evolving pictures collaboratively online. *Conference on Human Factors in Computing Systems - Proceedings*, 1759–1768. doi: 10.1145/1357054.1357328
- Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural content generation in games*. Switzerland: Springer International Publishing.
- Shalev-Shwartz, S. (2011). Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2), 107–194. doi: 10.1561/22000000018
- Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., . . . Wu, Y. (2018). Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2018-April*, 4779–4783. Retrieved from <http://arxiv.org/abs/1712.05884> doi: 10.1109/ICASSP.2018.8461368
- Shu, R., Bui, H. H., Zhao, S., Kochenderfer, M. J., & Ermon, S. (2018). Amortized inference regularization. *Advances in Neural Information Processing Systems, 2018-Decem(NeurIPS)*, 4393–4402.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. Retrieved from <http://www.nature.com/doifinder/10.1038/nature16961> doi: 10.1038/nature16961
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., . . . Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 1140–1144. doi: 10.1126/science.aar6404
- Simon, J. (2019). *artbreeder.com*. Retrieved from <https://artbreeder.com/>
- Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv preprint arXiv:1312.6034*, 1–8. Retrieved from <http://arxiv.org/abs/1312.6034>
- Sims, K. (1994). Evolving virtual creatures. *Siggraph '94, SIGGRAPH '(July)*, 15–22. Retrieved from <http://dl.acm.org/citation.cfm?id=192167%5Cnhttp://portal.acm.org/citation.cfm?doid=192161.192167> doi: 10.1145/192161.192167
- Smith, G., & Leymarie, F. F. (2017). The Machine as Artist: An Introduction. *Arts*, 6(4), 5. doi: 10.3390/arts6020005
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014).

- Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1), 1929–1958.
- Stanley, K. O., & Lehman, J. (2015). *Why greatness cannot be planned: The myth of the objective* (Springer, Ed.). doi: 10.1007/978-3-319-15524-1
- Sturm, B. (2015). *Recurrent Neural Networks for Folk Music Generation*. Retrieved from <https://highnoongmt.wordpress.com/2015/05/22/lisls-stis-recurrent-neural-networks-for-folk-music-generation>
- Sutskever, I. (2013). *Training Recurrent neural Networks* (Unpublished doctoral dissertation). University of Toronto.
- Sutskever, I., Martens, J., & Hinton, G. (2011). Generating Text with Recurrent Neural Networks. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 1017–1024).
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in neural information processing systems (nips)* (pp. 3104–3112).
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. *arXiv preprint arXiv:1512.00567*.
- Theis, L., van den Oord, A., & Bethge, M. (2016). A note on the evaluation of generative models. *ICLR 2016 A*.
- Todd, S., & Latham, W. (1992). *Evolutionary art and computers*. Academic Press, Inc.
- Touvron, H., Vedaldi, A., Douze, M., & Jégou, H. (2019). Fixing the train-test resolution discrepancy: FixEfficientNe. *Advances in Neural Information Processing Systems*, 32, 12–16.
- Tresset, P., & Deussen, O. (2014). Artistically Skilled Embodied Agents. (April).
- Turing, A. (1948). *Intelligent Machinery, A Heretical Theory* (Tech. Rep.). National Physical Laboratory. Retrieved from http://www.alanturing.net/intelligent_machinery/
- Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, 59(236), 433–460. Retrieved from <papers2://publication/uuid/E74CAAC6-F3DD-47E7-AEA6-5FB511730877> doi: http://dx.doi.org/10.1007/978-1-4020-6710-5_3
- Ulm, M. (n.d.). *Das Alter des Löwenmenschen*. Retrieved from http://www.loewenmensch.de/figur_3.html
- Van Den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural discrete representation learning. *Advances in Neural Information Processing Systems, 2017-Decem(Nips)*, 6307–6316. Retrieved from <http://arxiv.org/abs/1711.00937>
- van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., . . . Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499*.
- van den Oord, A., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. *arXiv preprint arXiv:1601.06759*.
- van den Oord, A., Kalchbrenner, N., Vinyals, O., Espenholt, L., Graves, A., & Kavukcuoglu,

- K. (2016). Conditional Image Generation with PixelCNN Decoders. *arXiv preprint arXiv:1606.05328*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 2017-Decem*, 5999–6009. Retrieved from <https://zhuanlan.zhihu.com/p/34781297>
- Verma, P., & Smith, J. O. (2017). Neural Style Transfer for Audio Spectrograms. *Machine Learning for Creativity and Design - NIPS 2017 Workshop(Nips)*, 6–8.
- Veton Kėpuska, & Bohouta, G. (2017). Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx). *International Journal of Engineering Research and Applications, 07(03)*, 20–24. doi: 10.9790/9622-0703022024
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature(575)*, 350–354.
- Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2016). Matching networks for one shot learning. *Advances in Neural Information Processing Systems(Nips)*, 3637–3645.
- Wang, S., Shao, M., & Fu, Y. (2014). Attractive or Not ? Beauty Prediction with Attractiveness-Aware Encoders and Robust Late Fusion. *ACM MM*, 2–5. doi: 10.1145/2647868.2654986
- Wang, T. C., Liu, M. Y., Zhu, J. Y., Tao, A., Kautz, J., & Catanzaro, B. (2018). High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 8798–8807. doi: 10.1109/CVPR.2018.00917
- Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing, 13(4)*, 600–612. doi: 10.1109/TIP.2003.819861
- Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE, 78(10)*, 1559–1560.
- White, T. (2016a). Sampling Generative Networks. *arXiv preprint arXiv:1609.04468*. Retrieved from <http://arxiv.org/abs/1609.04468>
- White, T. (2016b). *Smile Vector*. Retrieved from <https://twitter.com/smilevector>
- Wiener, N. (1948). *Cybernetics or control and communication in the animal and the machine*. Technology Press.
- Wright, M., & Freed, A. (1997). Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 international computer music conference (icmc)*.
- Wu, J., Zhang, C., Xue, T., Freeman, W. T., & Tenenbaum, J. B. (2016). Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. (Nips). Retrieved from <http://arxiv.org/abs/1610.07584>
- Wu, Z., & King, S. (2016). Investigating Gated Recurrent Neural Networks for Speech Synthesis. In *Ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5140–5144). IEEE.
- Wu, Z., & Song, S. (2015). 3D ShapeNets : A Deep Representation for Volumetric Shapes.

- IEEE Conference on Computer Vision and Pattern Recognition (CVPR2015)*, 1–9. doi: 10.1109/CVPR.2015.7298801
- Xian, Y., Lampert, C. H., Schiele, B., & Akata, Z. (2018). Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–14. doi: 10.1109/TPAMI.2018.2857768
- Xie, S., & Tu, Z. (2017). Holistically-Nested Edge Detection. *International Journal of Computer Vision*, 125(1-3), 3–18. doi: 10.1007/s11263-017-1004-z
- Yang, J., Anishchenko, I., Park, H., Peng, Z., Ovchinnikov, S., & Baker, D. (2020). Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences of the United States of America*, 117(3), 1496–1503. doi: 10.1073/pnas.1914677117
- Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. *Icrl(2013)*, 1–8. Retrieved from <http://arxiv.org/abs/1409.2329>
- Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. *arXiv preprint arXiv:1311.2901*.
- Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2015). Loss Functions for Neural Networks for Image Processing. *arXiv preprint arXiv:1511.08861*.
- Zhao, S., Song, J., & Ermon, S. (2017). InfoVAE: Information Maximizing Variational Autoencoders. Retrieved from <http://arxiv.org/abs/1706.02262>
- Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *Proceedings of the IEEE International Conference on Computer Vision, 2017-October*, 2242–2251. doi: 10.1109/ICCV.2017.244
- Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710. Retrieved from http://openaccess.thecvf.com/content_cvpr_2018/papers/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.pdf