Colour for the Advancement of Deep Learning in Computer Vision

By

ASEI AKANUMA

Supervisors:

Prof. J. Mark Bishop and Dr. John Howroyd

A thesis submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

Department of Computing

Goldsmiths College

University of London

New Cross, London SE14 6NW, UK.

Supervisors:

Prof. J. Mark Bishop and Dr. John Howroyd

Declaration

Declaration of Authorship: I Asei Akanuma 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:

Abstract

This thesis explores several research areas for Deep Learning related to computer vision concerning colours. First, this thesis considers the one of the most long standing challenge that has remained for Deep Learning which is, how can Deep Learning algorithms learn successfully without using human annotated data? To that end, this thesis examines using colours in images to learn meaningful representations of vision as a substitute for learning from hand-annotated data. Second, is another related topic to the previous, which is the application of Deep Learning to automate the complex graphics task of image colourisation, which is the process of adding colours to black and white images. Third, this thesis explores colour spaces and how the representations of colours in images affect the performance in Deep Learning models.

Acknowledgements

This thesis would not have been possible without the support and valuable discussions I had with a number of people. To these people I owe my greatest thanks.

Firstly, and most importantly, my greatest appreciation goes to my Primary Supervisor Prof. Mark Bishop for his guidance, inspiration and mentorship throughout the years. Without your support I would not have been able to reach this far in my academic journey.

Secondly, I would like to sincerely thank my Second Supervisor Dr. John Howroyd for his technical expertise and advice, particularly in the early to mid stages of constructing this thesis, as well as his encouragement throughout. Your time and patience are greatly appreciated.

I would like to thank the all the current and former members of TCIDA for the support, and for the many discussions and comments at the supervision meetings which were often helpful. I would also like to thank my peers at the Goldsmiths' postgraduate programme, and especially Hooman Oroojeni for the many valuable (and often lengthy) discussions we had about Deep Learning which were most useful.

I especially need to thank my family, my parents and my two brothers for their unconditional love and support during what was at times a difficult journey.

Lastly, but by far from least I would like to thank all my friends for their patience and understanding. You are all fantastic.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

This thesis explores several research areas of Deep Learning related to computer vision concerning colours.

First, this thesis tackles the problem of self supervised learning which involves solving some predictive pretext tasks in which the inputs and labels can be both derived from the raw image data, thus requiring no manual labels. It has been previously hypothesised that image colourisation (i.e., predicting the colours in images) can serve as a reasonable task for self supervised learning. However, applying this in practice has found to be difficult due to the lack of generality from learning such an ad-hoc task. This motivated the investigation of how the image colourisation task can combined with another self supervised learning method which do not rely on a solving predictive tasks for images, and whether there are benefits to this new method.

Second, is a topic closely related to the first, which is the application of Deep Learning for solving the colourisation task itself, a complex graphics task which involves predicting plausible coloured versions of an image given just the lightness of that image. Previous systems which achieve visually impressive results for this task usually do so with a significant amount of computational cost. Streamlining architectural designs for computation efficiency would make the model more accessible, and available to devices which are computationally resource constrained.

Third, is the exploration of colour spaces in relation to Deep Learning Convolutional Neural Network models. For most Deep Learning models, it is standard for input images to be represented

in RGB colour space, despite the numerous other colour spaces in existence. It is therefore a benefit to know the effects of colour spaces on Deep Convolutional models and whether there are benefits of using one colour space over anther, or some combinations of them.

1.2 Thesis Overview

An overview of the thesis by chapter is given as follows:

- **Chapter 2** provides a literature review for Artificial Neural Networks. This chapter begins with a definition of machine learning. Afterwards, a historical context and review of various Artificial Neural Networks including Convolutional Neural Networks are provided. Following that, a few unsupervised learning methods for for Deep Learning is discussed. Lastly, a brief summary on colour concepts is given, and in particular, some mathematics for transforming colour spaces is described.

- **Chapter 3** explores how the colours in images can be used for self supervised representation learning in the visual domain. This chapter first provides a brief review of works in the literature which relate to self supervised learning while putting them into context. Two self supervised learning methods which involve the predictive task of image colourisation is then investigated leading to an implementation for one of the method, which leads to the finding that in practice, using colour predictions alone as a predictive task for self supervised learning is not sufficient, because the representations learnt by them do not generalise well enough to other tasks. Subsequently, this motivates the investigation of other methods for visual representation learning, in particular a method known as contrastive learning. This lead to the development of a self supervised learning method which combines both the colour prediction task with the contrastive learning method which delivered some promising empirical results for an evaluation of visual representation quality.

- **Chapter 4** applies Deep Learning for the graphics task of image colourisation. This chapter first begins by providing a brief survey for the literature of related works which uses

Convolutional Neural Networks for image colourisation where it is found that limitations of the conventional convolution layers hinders the quality of colourisation results. This chapter later builds on insights from state of the art Deep Learning models for image synthesis with the aim of improving colourisation quality. This work demonstrates that a task as complex as image colourisation can be achieved with remarkably low computation cost while producing visually good results by developing an image colourisation model which uses a streamlined architecture, with efficient factorised convolution layers and convolutional attention-mechanism layers.

- **Chapter 5** originally sought to find if performance gains are available from using (as inputs) different colour spaces for Convolutional Neural Network models trained for image classification. Through experiments which compares combinations of different colour spaces and convolutional models, it is shown that the use of different colour spaces alone does not seem to yield better performing Convolutional Neural Network models. However, through the use of self-attention mechanisms, an emergent patterns of internal activations in Convolutional Neural Networks is discovered.

- **Chapter 6** concludes this thesis with a summary and provide some directions for future works.

1.3 Contribution

The expected contributions of this thesis are: a novel self supervised learning method which combines both contrastive and predictive learning by solving them both jointly, using a multi-task Convolutional Neural Network architecture which is presented in chapter 3, and a discovered patterns of network activations in Convolutional Neural Networks using attention-mechanisms presented in chapter 5.

1.4 Sources of Data and Tools used in this Study

The **sources of data** used in this study is listed below:

- **Cifar10** and **Cifar100** [52] CIFAR10 is a popular benchmarking dataset for image recognition algorithms. It consists of 60000 colour images of 10 class objects which are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. They are in 32x32 pixel resolution. 50000 images are for training and 10000 are used as test images. Figure: 1.1 shows a few examples of the images. CIFAR100 is similar to CIFAR-10 except it has a more extensive list of class objects for which there are 100 in total.

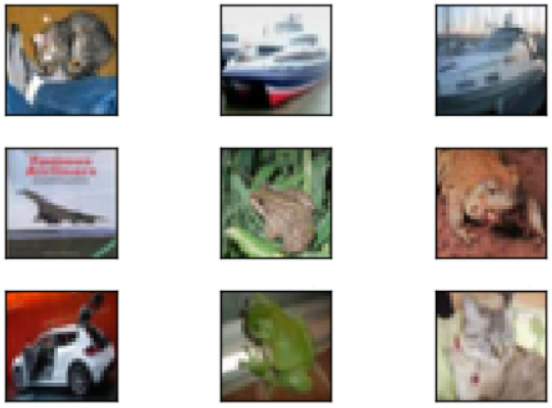


Figure 1.1. A few sample images from Cifar10

- **STL10** [18] STL10 is a dataset for image recognition which was inspired by the CIFAR-10 dataset. STL10 is a benchmark dataset that focuses on unsupervised feature learning. The images are in 96x96 pixel resolution and were taken from the ImageNet dataset. There are 10 classes which are: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. 5000 total images are provided as training set while 8000 images are provided to be used as test set. An additional 100,000 unlabelled images are provided as for unsupervised learning which include images which are different from the 10 classes in the train set. Figure: 1.2 shows a few examples of the images.

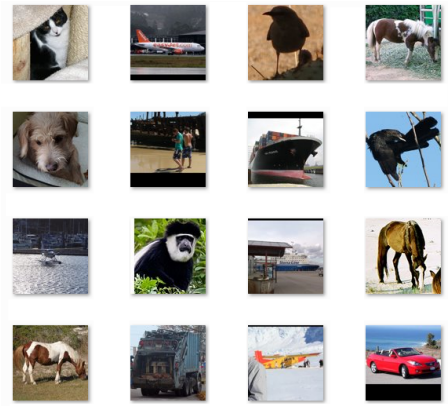


Figure 1.2. A few sample images from STL10

- **ImageNet** [23] is an extremely large visual database of photographs annotated by humans and is often considered the benchmark for the state of the art in computer vision. Annual competitions were held between 2010 and 2017 known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which involved challenging tasks using subsets of the database. ImageNet is further discussed in Section: 2.8.2

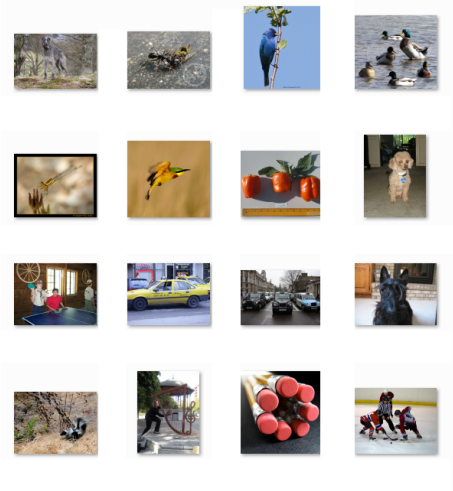


Figure 1.3. A few sample images from ImageNet

Unless stated otherwise, the experiments in this thesis was conducted using the Python programming language and the Tensorflow [1] library.

Chapter 2

Literature Survey: Artificial Neural Networks and Deep Learning

2.1 Chapter Overview

The aim of this chapter is to provide a literature survey for Artificial Neural Networks (Deep Learning) and a brief introduction to colour concepts and colour spaces. This chapter begins by first giving some context with an introduction to the sub-field of Artificial Intelligence (AI) known as Machine Learning. This is followed by a brief historical account of research in Artificial Neural Networks. Thereafter, a brief summary is given for the most basic type of the neural network known as the Multi-layer perceptrons. A special kind of neural network is then discussed in the context of computer vision known as Convolutional Neural Networks *considered the Deep Learning approach*, which is followed by a review of the more advanced methods to perform fine-grained predictions for images. Following that, some unsupervised learning approaches for Deep Learning are described and a recent trend in Deep Learning which involves using attention is discussed. Lastly, this chapter provides a brief background to colour concepts and colour spaces which are relevant to later chapters.

2.2 Introduction to Machine Learning

Machine Learning is a sub-field of AI which concerns the study of designing machines (hardware or software) which are able to learn from data. A widely-cited definition in the machine learning

literature was given by Mitchell [64] which states:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"

Given this broad definition, consider using a machine learning algorithm for the computer vision task of object recognition. Experience E can be the experience of observing a set of examples from a dataset containing images in a dataset matrix $X \in \mathbb{R}^{m \times n}$, where each m row of X represents an example described by n features. An example in X would therefore be an image, where each feature is a pixel value in that image. Additionally the experience E would also include observing the labels of the images given by a vector $y \in \{1, ..., k\}^m$, where the element y_i specifies which of the k object classes the image i belongs to. The task T then, for object recognition is to output a function $f : \mathbb{R}^n \rightarrow \{1, ...k\}$, where $f(x)$ could be interpreted as the machine learning algorithm's estimate for which object category the image x belongs to. The performance measure P would be the classification accuracy of the algorithm, in other words the frequency in which $f(X_i) = y_i$.

2.3 Machine Learning Tasks

Machine Learning tasks can be broadly categorised into three types which are: Supervised Learning, Unsupervised Learning, and Reinforcement Learning[1]. Among these, this study is primarily interested in the Supervised and Unsupervised Learning. The following gives a brief explanation for each of these and describes their differences.

2.3.1 Supervised Learning

Supervised learning problems involves learning from examples that have labels supplied in advance. A common task in supervised learning is the Classification problem, where there the task is to learn

[1] Although outside of the scope of this study, Reinforcement Learning is undeniably a popular topic which has been extensively studied in Machine Learning, for an excellent resource readers are referred to [85]

a function $f(x)$ that maps examples to a category y. An important distinguishing characteristic of the supervised learning is that it requires manually labelled examples for data.

2.3.2 Unsupervised Learning

Unsupervised Learning refer to machine learning tasks where labels y are not supplied, hence only the examples X are provided. The purpose of unsupervised learning to discover some structure from data, which can be some representation of important features. Common examples of unsupervised learning include: clustering, dimensionality reduction and generative models. Unsupervised learning is further discussed in Section 2.10.

2.4 A Brief Historical Context: Artificial Neural Networks

Artificial Neural Networks (or neural networks for short) is a popular approach in machine learning which was inspired by the way the biological nervous systems such as the brain works. Research in Artificial Neural Networks originated from the work of McCulloch and Pitts in 1943 as they defined the first mathematical model for an artificial neural network [62]. This foundational work was then followed by Donald Hebb who proposed an explanation for the adaptation of neurons in the brain by suggesting the strengthening of the neural pathways each time they are used. Hence, this gave a theory for the learning mechanism of the brain. In his widely-cited book "The Organization of Behavior" Hebb states [37]:

"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Building on McCulloch and Pitts neural network model and Hebb's insight, Frank Rosenblantt began his work to develop a machine that modelled the eyes of a fly. This led to the development of a neural network model for visual pattern recognition known as the Perceptron in 1958 [74]. The Perceptrons were different from the work of McCulloch and Pitts as it was able to learn by using

an adaptive rule. During that period, Artificial Neural Networks gained significant popularity in research.

However, in 1969 Minsky and Papert pointed out certain limitations of Rosenblatt's Perceptron models. In their monograph "Perceptrons" [63], they showed the Perceptron's inability to calculate non-linearly separable functions as well as being unable to discriminate between some simple properties of patterns such as the 'connectedness' [63]. For the above reasons coupled with the limitations of the available processing power at the time, neural network research fell out of favour which resulted in reduced interest and funding, even bringing it to a complete halt in the US for about 10 years [3].

Renewed interest in neural networks began in the mid 1980's with the work of Hopfield who described a network with associative memory using statistical mechanics and popularised a neural network model known as the Hopfield networks to solve optimization problems [42]. This was followed by the works of Rumelhart, Hinton and Williams who announced the discovery of a method that allowed a multi-layer feed forward networks to learn to discriminate between non-linearly separable classes using a method known as the "back-propagation of errors" [77, 76] which was discovered by Werbos in his 1974 Ph.D. dissertation [94].

The next breakthrough in the neural network research came in 2006 when Hinton and Salakhutdinov [38] successfully trained a deep feedforward neural network by training each layer at a time in a greedy layer-wise fashion and stacking layers on top of each other to generate a deep network, which were then fine-tuned using the back-propagation. The result was an initial resurgence of interest in academic research.

In 2012 a Convolutional Neural Network designed by Alexander Krizhevsky known as the AlexNet [53] was introduced in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [23, 78]. The AlexNet went on to achieve a 15.3 percent top-5 prediction error outperforming all competing methods at the time by an unprecedented margin. This breakthrough by Krizhevsky *et al.* [53] is often credited for the rise in popularity of the method known as *Deep Learning* [57, 30] which has attracted significant academic interest and demand for commercial applications as they provide

an approach for neural networks to achieve highly-scalable learning for large-scale data.

What comes next? Despite the substantial success garnered by Deep Learning, challenges still remain, because at present most mainstream approaches of Deep Learning rely on Supervised learning which require large dataset that have been hand-annotated, which clearly has its limitations when considering the scalability of this approach. Thus, solving the long-standing problem of making Deep Learning algorithms learn without supervision from human annotated data will likely prove to be a next significant step in development for this field.

2.5 The Multilayer Perceptron

This section serves as an introduction to Artificial Neural Networks. In particular, the following describes the most basic kind of Artificial neural network known as the feed-forward Multilayer Perceptron.

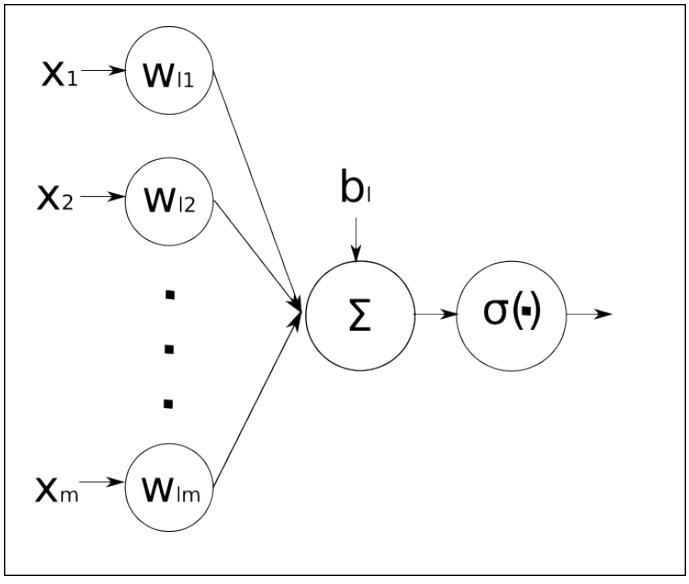


Figure 2.1. An artificial neuron

2.5.1 Basic Structure

Fundamentally, an Artificial neural network consists of basic computational elements which are somewhat analogous to biological neurons[2], these are variously (and often interchangeably) referred to as the artificial neurons, units or nodes. The units have internal parameters, which comprises its weights and a bias, which can be considered their synaptic weights, these are the parameters that are adapted during learning. Figure 2.1 shows a basic structure of a unit where $x_1, x_2, x_3, ...x_m$ are input signals, $w_{l1}, w_{l2}, ...w_{lm}$ are the weights and b_l is a bias. A unit receives inputs and computes the output of an activation function $\sigma(\cdot)$ for the linear combination of the inputs and the bias. The added bias has an effect of applying an affine transformation. The activation of a neuron is non-linear and can be stated as:

$$a_l = \sum_{i=1}^{m} w_{li} x_i + b_l \tag{2.1}$$

$$z_l = \sigma(a_l) \tag{2.2}$$

Multilayer perceptrons are usually organized into a hierarchical, layered structures where the overall structure can contain multiple layers and each layer can contain one more or units. In neural network literatures, *the network architecture* is used to specify the number of neurons in each layer, the depth (the number of layers) and the connectivity pattern of the weights in the neural network.

Figure 2.2 shows an example of a basic architecture structure for the Multilayer Perceptron which consists of an input layer, a hidden layer and an output layer. In this architecture, the network is assumed to be fully connected i.e, units between the adjacent layers are all connected by the

[2]Although it should be noted that most modern artificial neural network systems have been highly abstracted from their original biological inspirations, in fact it is unapt to say modern Artificial Neural Networks resemble biological ones.

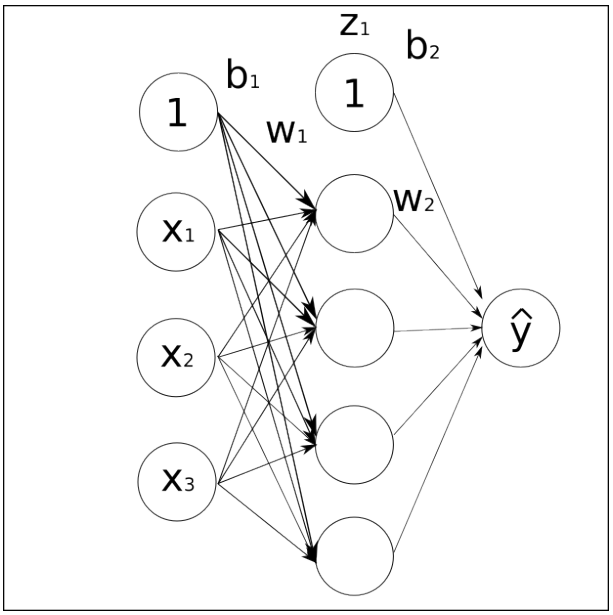


Figure 2.2. A basic feed-forward Multilayer perceptron

weights. In addition, Artificial Neural Networks are often viewed as computational graphs[3]. It is worth noting that neural networks such as Multilayer perceptrons which have acyclic computational graphs belong to a broad class of models called the feed-forward neural networks[4].

2.5.2 Activation functions

The activation functions are used to add non-linearities between layers. The following gives an overview of the different kinds of the commonly used activation functions used in the hidden layers.

The Logistic Sigmoid was traditionally a very popular activation function for neural networks which transforms an input into a value between 0 and 1.

$$\text{sigma}(a) = \frac{1}{1 + exp(-a)}, \tag{2.3}$$

[3]A computational graph is a representation of composite functions as a network of connected nodes, where each node is an operation or a function.

[4]Another class of the neural networks is the Recurrent neural networks which have cyclic weight connections, though these are outside the scope of this study.

$$\text{sigma}'(a) = \text{sigma}(a)(1 - \text{sigma}(a)), \tag{2.4}$$

The Hyperbolic tangent (Tanh), has a similar shape to the Logistic Sigmoid and is a function that outputs values between -1 and 1.

$$\tanh(a) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.5}$$

$$\tanh'(a) = 1 - \tanh^2(a). \tag{2.6}$$

The Rectifier is a non-linear function that returns the input value input directly, or the value 0 if the input value is 0 or less.

$$\text{relu}(a) = max(0, a), \tag{2.7}$$

$$\text{relu}'(a) = 1, \text{ if } a > 0; \ 0, \text{ otherwise.} \tag{2.8}$$

The Softmax activation which is described below is another kind of activation function used in the final layer for classification models. It is used to normalise the previous hidden layer (the logits), which has K units (a vector with K-dimensional real values), into classification scores for each class of K (a K-dimensional vector with real values in the range of 0 to 1 which sum to 1). Given the previous layer $a = [a_1, a_2, ..a_K]$ the softmax is given as:

$$\text{softmax}(a) = \frac{exp(a)}{\sum_k exp(a_k)}, \tag{2.9}$$

Note that the classification prediction of a network is given by the class of the unit that has the

highest score.

2.6 The Gradient Descent Algorithm

Algorithm 1 Gradient Descent

Input: loss function ε, learning rate η, dataset X, y and model $\mathcal{F}(\theta, x)$
Output: Optimum parameters θ which minimizes ε

1: **repeat**
2: $\quad \hat{y} = \mathcal{F}(\theta, x)$;
3: $\quad \theta = \theta - \eta \cdot \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \varepsilon(y, \hat{y})}{\partial \theta}$
4: **until** Convergence

The most popular approach for optimising a neural network is the Gradient decent algorithm [77]. Gradient descent is an optimisation method that finds a local minima for a given objective. The objective are in a form of a function called the loss function ε, which gives a quantitative measure (loss) of the neural network performance based on the network output $\hat{y}$ and target y. The gradient descent can be summarised as an algorithm that makes small changes to the neural network parameters along the gradient of errors towards a minimum error value until some convergence to a solution is reached by iteratively taking small steps of size η called the learning rate. Algorithm 1 gives an overview of the gradient descent. It should be noted that there have been various improvements introduced over the years for gradient descent (for details see Section 2.8.3).

2.6.1 The Stochastic Gradient Descent

Alternatively, the stochastic gradient decent (or some time refereed to as mini-batch gradient descent) is described in Algorithm 2. Notice that unlike the standard gradient decent, the algorithm additionally iterates over fewer batches over the whole dataset, where one cycle of the whole dataset is called an epoch. Stochastic gradient descent is often used when it is impractical to compute updates over the whole dataset at once. The choice for the size of the batches (batch size) are often determined by factors such as hardware memory requirements during implementa-

Algorithm 2 Stochastic Gradient Descent

Input: loss function ε, learning rate η, dataset X, y and model $\mathcal{F}(\theta, x)$
Output: Optimum parameters θ which minimizes ε

repeat
  Shuffle X, y
  for each batch of x_i, y_i in X, y **do**
    $\hat{y} = \mathcal{F}(\theta, x_i)$;
    $\theta = \theta - \eta \cdot \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \varepsilon(y, \hat{y})}{\partial \theta}$
  end for
until Convergence

tion although in general, larger batch sizes provides a better quality of the approximation for the update direction because the gradients are averaged over more examples from the dataset.

2.6.2 Back-propagation

Algorithm 3 Back-propagation

Input: A Neural Network with l layers, activation function σ_l, output of hidden layer $h_l = \sigma_l(W_l^T h_{l-1} + b_l)$ and network output $\hat{y} = h_l$
Compute the gradients $\delta \leftarrow \frac{\partial \varepsilon(y_i, \hat{y_i})}{\partial y}$

for $i \leftarrow l$ to 0 **do**
  Calculate the Gradients for the current layer:

$$\frac{\partial \varepsilon(y, \hat{y})}{\partial W_l} = \frac{\partial \varepsilon(y, \hat{y})}{\partial h_l} \frac{\partial h_l}{\partial W_l} = \delta \frac{\partial h_l}{\partial W_l}$$

$$\frac{\partial \varepsilon(y, \hat{y})}{\partial b_l} = \frac{\partial \varepsilon(y, \hat{y})}{\partial h_l} \frac{\partial h_l}{\partial b_l} = \delta \frac{\partial h_l}{\partial b_l}$$

  Apply gradient descent using $\frac{\partial \varepsilon(y, \hat{y})}{\partial W_l}$ and $\frac{\partial \varepsilon(y, \hat{y})}{\partial b_l}$
  Back-propagate gradient to the lower layer

$$\delta \leftarrow \frac{\partial \varepsilon(y, \hat{y})}{\partial h_l} \frac{\partial h_l}{\partial h_{l-1}} = \delta \frac{\partial h_l}{\partial h_{l-1}}$$

end for

Back-propagation is the algorithm used with gradient descent to optimize the weight parameters. Back-propagation involves repeating the *chain rule in calculus* to efficiently calculate the gradient from the last layer to the first layer in the network as shown in Algorithm 3. To illustrate the

relevance of the chain rule, in back-propagation consider the following case where a network has two layers, i.e., $l = 2$, the function can then be expressed as:

$$\hat{y} = f(x) = f(g(x)) \tag{2.10}$$

in the above case the chain rule derives the above function as:

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial f(x)}{\partial x} = f'(g(x)) \cdot g'(x) \tag{2.11}$$

2.6.3 Loss Functions

In (multinomial) **classification problems** such as object recognition, the labels are often converted to be in the form of a 1-of-K (one-hot) encoding[5] A common loss function for the classification model is the categorical cross-entropy loss which is as follows:

$$CE = -\sum_{i}^{K} y_i log(\hat{y}_i) \tag{2.12}$$

where K is the number of classes, y_i and $\hat{y}_i$ are the labels and the outputs of the softmax layer (with the scores for each class i in K) respectively. For **regression problems** where the target variables are continuous real values i.e, $y = \mathbb{R}^m$, the commonly used loss function is the Mean squared error loss stated as:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.13}$$

It is worth noting that for more complex problems can require more elaborate loss functions. In addition loss functions can be further hand engineered in which can be some variant of the above

[5] a binary vector where of all elements are 0s except the index k which is set to 1 to indicate the class

functions or a more complex function.

2.7 Training a Neural Network In Practice

This section provides some discussion for the more practical aspects of training a neural network. These include the network initialisation, learning rate, regularisation, and the model selection process all which all require careful considerations when training a neural network.

2.7.1 Network Initialisation

Before the neural network training can begin, the neural network parameters (weights and biases) must first be initialised. Crucially, the initialized weight must not be symmetrical, instead weights are initially assigned random weights as initializing the weights uniformly leads to the weights being updated uniformly during training due to the back-propagation process, resulting in the network not learning features useful for the task. Therefore, the randomness is used to break the symmetry in the parameters of the network. The random initialization is usually applied by sampling from a uniform or Gaussian distribution. Initialising a neural network parameters requires careful consideration because if the initial parameters values are too small, it may lead to slow learning, while values too large could lead to divergence from a solution. Studies have since proposed best practices for the neural network initializations [83]. One of the most popular and successful approach for the network initialisation includes the Xavier initialisation [29] which suggest that for every layer in l the parameters should be initialized as follows:

$$W^{[l]} \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{(l-1)}}), b^l = 0 \tag{2.14}$$

Where $n^{(l-1)}$ is the number of units in the previous layer. Thus, all biases are initialised as 0 and all weights are sampled from a normal distribution where the mean is 0 and the variance is $\sigma^2 = \frac{1}{n^{(l-1)}}$.

2.7.2 Learning Rate

The learning rate is the step size for each update made in the gradient decent algorithm (see Section 2.6.1). Setting an appropriate learning rate is perhaps one of the most important aspect of training a neural network with gradient decent. This is because similar to the network initialization, choosing a values for η that is too small will result in slow training and a long time to converge. Conversely, a learning rate that is too large could potentially diverge from a solution. Since there is no way of knowing what an optimal learning rate is beforehand for a given task, it is common practice is to start with values such as $\eta = 0.1$ and adjust the learning rate over multiple trials of training. It is also often the case that the a single constant learning rate is not optimal for the whole training process, thus requiring the learning rate to be reduced at certain stages of the training, which can be implemented by using a learning rate schedule. A learning rate training schedule can be implemented by reducing learning rates after a certain number of iterations (or epochs) have been completed, in other words decaying the learning rate of an initial learning rate, set at the beginning of the training. The rate of the reduction in learning rate could be at some constant scheduled rate, or by some other factor such as an exponential rate.

2.7.3 Momentum

Momentum [70] and the other proposed variants [86] are methods to help gradient descent accelerate in relevant directions. Momentum works by adding a fraction γ to the update vector of the past step to the current update, which help navigating ravines areas of the error surface to allow for faster convergence. A gradient decent with momentum could be stated as:

$$v_t = \gamma v_{t-1} + \eta \nabla \mathcal{F}(\theta) \tag{2.15}$$

$$\theta = \theta - v_t \tag{2.16}$$

The momentum term γ is therefore, another hyper-parameter. A common practice is to set $\gamma = 0.9$.

2.7.4 Regularisation

It is important to consider that the ultimate objective of a trained neural network model is to predict new examples, i.e., they must generalise to data not seen during the training process. A trained model that performs well on the examples it is trained on, but are unable to perform well on un-seen examples are undesirable and are often an indication of what is known as *model overfitting*. There are several methods for dealing with the model overfitting problem, such as the weight decay method, dropout regularisation (see Section 2.8.3). Below provides details for the most simple approach known as the weight decay.

The weight decay method prevents the models from overfitting by penalising large weight parameters during the model training which controls the complexity of the models. L2 regularisation is a weight decay method based on the L2 norm which is simply added to the loss objective. For a model $\mathcal{F}(\theta, x)$ the L2 regularisation can be stated as:

$$\hat{\varepsilon}(\mathcal{F}(\theta, x), y) = \varepsilon(\mathcal{F}(\theta, x), y) + \lambda\Omega_{l2}(W) \tag{2.17}$$

where $\Omega_{l2}(W) = \|W\|^2$, (the L2 norm of all the weight parameters in the model) and λ is a hyper-parameter which controls the regularisation strength, which is usually set to a small values such as 0.0001.

2.7.5 Generalisation and Model Selection

As emphasised above, a trained neural network model must generalise to data not seen during the training process. For this reason, the examples in the datasets are usually partitioned into smaller subsets of training, validation and test sets, where an important assumption known as the *i.i.d.* assumptions[6] must be made about the data. Models are then, directly optimised on the training

[6]These assumptions state that the data is independently and identically distributed and each example is drawn from the same distribution p_{data} [21] in other words $p_{data} = (X, y) = \prod_i p_{data}(X_i, y_i)$

set where the majority of examples in dataset resides. This training process is usually guided by evaluating the model's performance on the relatively smaller portion of examples which were set aside in the validation set, which could be used to optimise many of the hyper-parameters of the neural network such as the learning rate, momentum, regularisation values. Performance of the validation set can also reveal issues such as the model over-fitting by comparing training and validation set performance. A final model is selected by evaluating the model's performance on the test set which was unseen to the model throughout the training process.

2.8 Convolutional Neural Networks and Deep Learning

The aim of this section is to introduce the Convolutional Neural Network, and to provide a review of the recent developments and advanced techniques that have contributed to their success. Convolutional Neural Networks are a type of feed-forward neural network that are highly optimised for processing 2D inputs. They are most popularly used for computer vision applications, especially for tasks such as image classification. The some of the earliest Convolutional Neural Network architectures was proposed in the late 1980s [27, 58] though the limitations of the hardware available at the time made the implementation challenging and thus, the approach less popular. Notably it was the work of LeCun *et al.* [58] that proposed a gradient-based training method for the Convolutional Neural Network. Most modern Convolutional Neural Networks uses the gradient descent method to train network architectures which consists of many layers which are organised in a hierarchical structure, giving them depth. The depth of layers allows different levels of representations of the inputs to be learnt where typically the initial layers learn to detect some basic patterns such as edges while layers further in depth learns more complex patterns that are useful for solving tasks. This method of learning allows the automatic extraction of features in a highly scalable end-to-end manner. The Neural Networks of the kind described here are called *Deep Learning*.

Traditionally, machine learning approaches for visual recognition usually involved applying some feature extraction algorithms, which often required laborious hand engineering to design. These

traditional feature extraction algorithms such as Scale Invariant Feature Transform (SIFT) [61] and Histogram Oriented Gradient (HOG) [95]. These extracted features were then often exploited by other machine learning algorithms such as the Support Vector Machine [20] and Random Forest [41] to draw decision boundaries for the classification. However, these methods made the machine learning pipelines often overly complex and unwieldy for practical use.

The massive performance gains and time savings made by the end-to-end approach of Deep Learning as demonstrated by the works such as the AlexNet [53], when compared to the traditional machine learning approaches (with feature engineering) resulted in a paradigm shift in 2012. The Deep Learning approach have since proven to be widely successful which subsequently made them the de facto standard for visual recognition tasks.

2.8.1 Overview of Convolutional Neural Networks

The following provides details on the implementation of a Convolutional Neural Network. A Convolutional Neural Networks architecture mainly consists of three basic kinds of layers which are: the convolution, pooling and classification layers, which are described in the following.

Convolution layer

The most important component of the Convolutional Neural Network is the Convolution layer. The Convolution layer is based on the 2D convolution operations which can be described as taking an input I and sliding a kernel K of a fixed window size across the spacial dimension of the input and taking the dot product. The convolution operation[7] denoted by $*$ can be stated as:

[7]Speaking strictly, mathematically the operations performed below are actually cross-correlation operations i.e., 2D convolutions without flipping the kernels. In Deep Learning literatures the terms *convolution* and *cross-correlation* are sometimes used interchangeably since the theoretical implications are not very significant to the neural network implementations. For simplicity, and to remain consistent with the terminology of most Deep Learning literatures, the rest of this thesis will simply refer to them as convolutions.

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.18}$$

When convolutions are applied as layers in a Convolutional Neural Network, the input (for example, a 2D image) $\mathbf{x} \in \mathbb{R}^{H \times W' \times C'}$ is convolved with a number of learnable filters $\mathbf{w} = [w_1, w_2, ...w_C]$ (which acts as the convolution layer's weights) where w_c refers to the parameters of the c-th filter. The convolution layer's output is then $\mathbf{z} = [z_1, z_2, ..z_C]$, where:

$$z_c = \sigma(\sum_{s=1}^{C'} w_c^s * x_c + b_c) \tag{2.19}$$

where $w_c = [w_c^1, w_c^2, ..., w_c^C]$, $\mathbf{x} = [x^1, x^2, ..., x^{C'}]$ and $z_c =\in R^{H \times W}$. w_c^s is a 2d spatial kernel representing a single channel of w_c corresponding to $\mathbf{x}$, b_c is the bias term added to the cth filter and σ is a non-linear activation function. The input $\mathbf{x}$ to the first layer of a Convolutional Neural Network are the raw images, (usually in a 3-channel RGB format) while subsequent layers use the previous layer's output feature maps as their inputs.

The filter sizes (the spatial window size of the filters) called the *receptive field* are usually kept significantly smaller than the input image size. By having filters with spatial sizes that are smaller than the inputs, the convolutional layers allows for sparse connectivity and parameter sharing (by tied weights), which reduces the memory requirements of the models while improving efficiency. Furthermore, the parameter sharing of the convolutional layers gives them the property known as equivariance to translations [30].

Pooling layer

The pooling layers are another type of layer that are often added after convolution layers which involve a pooling function to further modify the outputs of layers. A popular pooling layer is the max-pooling layer which involves taking the maximum value within a rectangular neighbourhood of the inputs for a specified window size. Other pooling layers which uses variants for the pooling

functions also exist which includes the use of average or l2 norms pooling. Pooling helps to make the representation between layers invariant to small translations of the input, in addition to reducing the spatial dimensions of the layer's output.

Fully connected layer

Lastly, is the fully connected layer. The fully connected layer essentially the same as the layers described in the multilayer perception. The fully connected layer allows the 2D feature maps from the convolution or pooling layers to flattened into feature vector which are then used by the final task-specific layer e.g, a final classification softmax layer.

Strides and Padding

Note that the convolution and pooling layers described above results in a 2D spatial down-sampling of the inputs. The convolution and pooling layers described above can also be implemented with different stride sizes S. Varying the stride sizes with $S < 1$ causes the receptive field/window of the convolution/pooling layers to move at a faster rate resulting in an increase of the rate at which the spacial dimension are down sampled.

Conversely, paddings of 0 values can be inserted in the rows and columns of the outer boarders of the 2D inputs or feature maps to preserve more spacial dimensions between the convolutional or pooling layers. In the case where the 2d output dimensions needs to match the input dimensions, the amount of paddings P needed in the boarders can be determined by the following equation:

$$P = (F - 1)/2 \tag{2.20}$$

where F is the size receptive field/window size of the convolution or pooling layer and assuming the stride size is 1.

Parameters, Memory Requirements and Computational Cost

It is important to consider the complexity and memory footprint of the Convolutional Neural Network implementation. The complexity of the models are considered in terms of parameter and memory as this would dictate the hardware requirements of the model implementation. Firstly, it is important to consider is the size spatial dimension of the output feature maps which can be calculated as:

$$M = \frac{(N - F + 2P)}{S} + 1 \tag{2.21}$$

where N is the size of the inputs, F is the size of the filter, S is the stride size and M is the size the spacial dimension output feature map, assuming the inputs and kernel are a 2D square. The total number parameters for a layer l can then be calculated as:

$$\text{Param}_l = (F \times (F + 1) \times FM_{l-1}) \times FM_l \tag{2.22}$$

where $Param_l$ is the total number of parameters in the lth layer , FM_{l-1} is the number of input feature maps and FM_l is the number of output feature maps. Storing each parameter as a single precision floating-point allows 2.5×10^5 parameters to be stored per 1 Mb of memory space. Lastly, the amount of memory $mboxMemory_l$ needed for operations for the lth layer is calculated as:

$$\text{Memory}_l = (N_l \times N_l) \times FM_l \tag{2.23}$$

Lastly, the computational cost of the convolution is:

$$F \times F \times FM_{l-1} \times FM_l \times N \times N \tag{2.24}$$

Depthwise Separable Convolutions

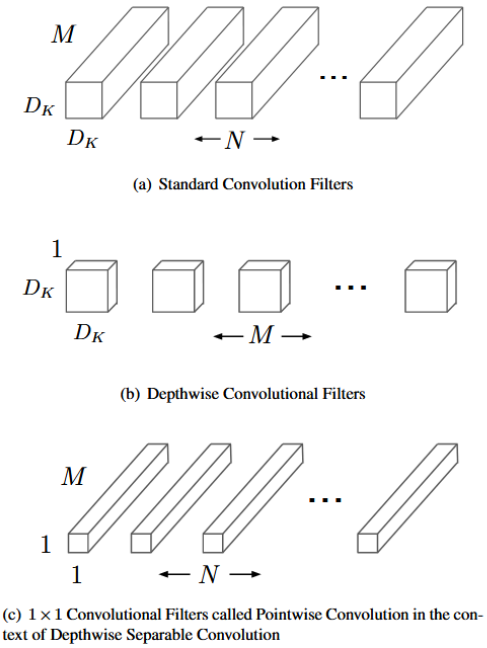


Figure 2.3. An illustration of the Depthwise Separable Convolution (Image source: [43])

More recently, a form of factorized convolutions known as the Depthwise separable convolution was introduced by [80], which offer substantial reduction in computational cost over the standard convolution layers described above. Depthwise separable convolutions work by spitting the standard convolution into two steps of factorized convolutions which results in two layers. These two steps are the depthwise convolution and the pointwise convolution. This process can be described as the following.

First, the Depthwise convolutions apply a single filter for each input channel which produces a number of M feature maps. Then, the pointwise convolutions apply a 1×1 convolution to create a linear combination of the output of the depthwise convolution outputs. Figure 2.3 illustrates this process, where the standard convolutional filters in (a) are replaced by two layers: the depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable convolution filter. The depthwise separable convolutions, offer a computational reduction over the standard convolution of:

$$\frac{1}{FM_l} + \frac{1}{F^2} \tag{2.25}$$

The depthwise separable convolutions were used by popular network architectures such as the Xception [16] and Mobilenets [43] which resulted in models with significantly reduced computational costs.

2.8.2 Convolutional Neural Network Architectures and ImageNet

Since the advent of Alexnet [53], the designs of Convolutional Neural Networks architectures have become a popular area of research. The advancements made in the architectural designs over recent years have resulted in Deep Learning methods making dramatic improvements in the state of the art in the field of computer vision, and have since achieved performances that seemingly surpass an estimated performance for humans [78].

Architecture	Year	Top 1 accuracy	Top 5 accuracy	Number of ∼parameters
AlexNet[53]	2012	63.3%	84.6%	60M
Inception-V1[87]	2013	69.8%	89.9%	5M
VGG-16[81]	2014	74.4%	91.9%	138M
Inception-V2[47]	2014	74.8%	92.2%	11.2M
ResNet-50[35]	2015	77.15%	93.29%	25.6M
DenseNet-121[45]	2016	78.54%	94.46%	20M
DPN-131[14]	2017	81.45%	95.84%	80M
MobileNet-224(L)[43]	2017	70.6%	89.5%	4.2M
ShuffleNet(L)[107]	2017	70.9%	89.8%	5.3M
Human Performance[78]	-	-	94.9%	-

Table 2.1. Popular Convolutional Neural Network architectures and their ImageNet accuracy performance

Such remarkable progress was perhaps not possible without the introduction of large-scale datasets. The most popular large-scale dataset is imagenet [23] introduced in 2009, which is a large database containing over 14 million images which were annotated by humans using crowd-sourcing services

such as the Amazons Mechanical Turk [78]. The imagenet project's purpose was to provide a large-scale ontology of images, the database is therefore organised according to the wordNet hierarchy[8]. The imageNet project included annual competitions which were held between 2010 and 2017 known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) where various tasks for image classification and object detection were set as challenges using subsets of the ImageNet database. Among them is the highly competitive benchmarking challenge of ImageNet-1K, which is a classification task of 1000 object categories where networks are evaluated for the accuracy of their top 1 and top 5 predictions. Table 2.1 provides a non-exhaustive list of some of the most notable architectures proposed over recent years and their imageNet-1K performance accuracy as well as an estimated performance of human attempts for the same task. Table 2.1 additionally also includes models which have resource efficient (lightweight) architectures, which by design have significantly less computational complexity and parameter requirements, giving them balance between performance and complexity. To separate these from the standard architecture, they are indicated with an (L) added next to their architecture name.

Notably, the general trend that is prevalent among more recent architecture designs is more interconnectedness between layers and improvements in the designs of the convolution layers for better efficiency. The following provides a brief summary of the various architectures which are essential to the discussions in the subsequent chapters in this thesis.

AlexNet (2012)

The **AlexNet** is a Convolutional Neural Network architecture that was proposed by Krizhevesky *et al.* which won the ILSVRC in 2012. AlexNet was considered one of the most remarkable breakthroughs in machine learning, when it outperformed all other traditional machine learning approaches for image classification by a large margin. It is in fact often credited for the recent rise in popularity in neural network research for computer vision and even beyond [4].

The AlexNet architecture has 60 million parameters with 7 layers of which, 5 are convolution

[8]The wordNet is a large lexical database of English: https://wordnet.princeton.edu/

layers, 2 are fully connected layers. The AlexNet was also the first to introduce the idea of Local Response Normalisation, which involves normalizing patches of the feature maps based on its neighbouring values, and the concept known as the dropout regularisation (see Section 2.8.3)

The Alexnet architecture has inputs for images of sizes 224×224. The first layer uses a convolution layer with 96 filters with receptive field of 11×11 followed by a maxpooling of 3×3 and Local Response Normalisation. The second layer repeats the same operations but with 256 filters and with 5×5 convolution. The third, fourth and fifth layers uses 3×3 convolutions and Relu non-linear activations, which each have filter sizes of 384, 384 and 256 respectively. The sixth and seventh layers are fully connected layer with 4096 units each, which is followed by a final softmax output layer to compute the classification of 1000 class scores. The whole network was trained using the dropout regularisation, which significantly reduced overfitting. It should also be noted that for the original implementation of Alexnet, a 2-stream network was used by halving the sizes of the parameter in each layers, which allowed training to be carried out on a machine with 2 GPUs.

VGGNets (2014)

The **VGG** models [81], named after the Visual Geometry Group are convolutional network architectures which have several variants. Their best model which was submitted to the ILSVRC competition in 2014 was the runner up (2nd place) that year. However the model's architecture has remained popular due to their innovative design and robust performance.

The main contribution of the VGG models was the demonstration of the importance of depth in visual representations as the VGG networks were considered very deep architectures at the time they were introduced. The variants of the architecture designs included models which were 16 and 19 layers in depth known as VGG-16 and VGG-19 respectively. The VGG network uses only 3×3 or 1×1 convolution filters throughout the whole network, unlike the many previous architecture designs which commonly used 11×11 or 7×7 convolution filters, as convolution with larger receptive fields have been found to be computationally expensive and require and high parameter

requirements. Subsequently, the VGG provided a more streamlined design for the convolution layers which allowed the architecture depth to be extended efficiently.

Most VGG networks typically uses a series of 3×3 convolution layers with relu non-linear activations. The filter numbers in the initial layer is 64 and are increased by a factor of 2 in every 2 or 3 proceeding layers where they are down-sampled by a pooling layer. This allowed the control over complexity and parameter requirements as the layers progress deeper. The series of convolution layers were then followed by 3 fully connected layers and a final sotfmax classification layer. Despite the various reduction in complexity the VGG networks are considered computationally expensive as models such as the VGG-16 has a 138 million parameter memory footprint.

ResNet (2015)

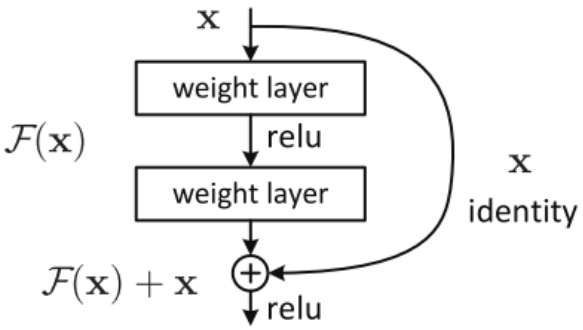


Figure 2.4. A Residual Block (Image source: [35])

Another breakthrough in the architecture design is the Residual Network or **ResNet** introduced by He *et al.* [35] which was the winner of ILSVRC in 2015. The ResNet introduced the concept of "identity shortcut connections" which are shortcut connections that skips over layers which promotes better gradient flow between layers and helps dealing with the problem known as vanishing gradient problem[9] [9]. ResNet also introduced modular structure to the Deep Learning architecture construction by introducing residual blocks as shown in Figure 2.4. The residual blocks which are stacked can compose models of varying depth. These residual block can be described as:

[9]Briefly, this is a problem that since back-propagation involves repeated multiplications, the gradient could become infinitely small for layers earlier in the network, which makes training deep architectures difficult.

$$y = \mathcal{F}(x, W_i) + x \tag{2.26}$$

where x and y are inputs and output vectors of the layer l respectively, $\mathcal{F}(x, W_i)$ represent the layer to be learned (the residual mapping), which are usually two layers of convolutions with a relu activation in between i.e., $\mathcal{F} = W_2\sigma(W_1x + b_1) + b_2$. The operation $\mathcal{F} + x$ is implemented by a shortcut connection and element-wise addition.

The ResNet design were able to successfully train models which are extremely deep without their performances degrading unlike previous model designs. Moreover, a refined version of the ResNet allowed the network depth to be extended even as much as 1000 layers deep [36]. The popular ResNet architecture inspired many other variants and modifications, these notable variants include wide (in channel depth) versions WideResNets [100] and versions which uses multiple branches which group convolutions which introduces cardinality in the Residual Block known as ResNeXt [96]. Furthermore, Densensets [45] and Fractalnets [54] are architecture designs which extend the ideas from ResNet.

DenseNet (2016)

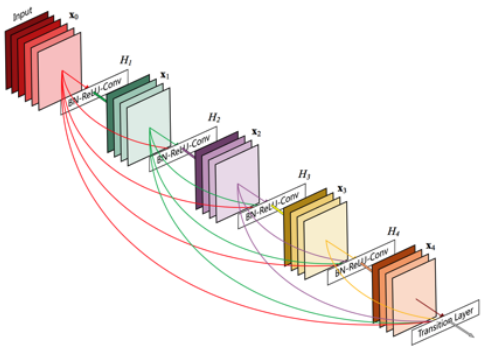


Figure 2.5. Layers in a DenseNet (A Dense Block) (Image source: [45])

The Densenet by Huang *et al.* [45] introduced a more densely connected architecture by concate-

nating each layers feature maps to the inputs (channel-wise) of every successive layers for groups of layers through a structure known as the Dense Block Shown in Figure 2.5. Specifically, a Dense Block can be described as:

$$x_l = H_l([x_0, x_1 ... x_{l-1}]) \tag{2.27}$$

x_l is the output of the lth layer, $[x_0, x_1 ... x_{l-1}]$ are the contented feature maps of the layers up to the $(l-1)$th layer and H_l is the operations performed at each layer within the Dense Block which includes batch-normilzation, a 3×3 convolution layer followed by a Relu activation. This increased inter-connectivity within the network encourages features to be reused which in turn, increases variation in the input of subsequent layers and improves efficiency of parameter use. The Densenets also include a structure known as the transition blocks which are inserted between the dense blocks, which each performs a 1×1 convolution and a batch normalisation followed by a 2×2 max pooling.

The Densenets achieves high performances despite using only a small number of filters per layer which vastly reduces the total number of parameters. The number of filers added at each layer in the Dense Block which the authors called the *growth rate* k thus, grows the number of filters at the constant rate at each proceeding layer. The growth rate are kept relatively small, configuration in the original work commonly used $k = 12$, $k = 24$ or even $k = 40$.

MobileNet (2017)

More recently, the Mobilenet [43] was introduced by Howard *et al.*, in 2017. The Mobilenets belong to a class of lightweight Convolutional Neural Network architectures (which include other architectures such as ShuffleNets) which aim to streamline the computations within the models which allow them to be embedded into devices with limited computational resources e.g. mobile devices. The MobileNet uses a series of 3×3 depthwise separable convolutions and batch normalisation. Subsequently this resulted in a reduction of between 8 to 9 times less computation than

the standard convolutions while only experiencing a small reduction in accuracy [43].

2.8.3 Advanced Techniques for Convolutional Neural Networks

This section describes some of the more modern and advanced techniques which have been introduced recently in the Deep Learning literature. Specifically, the following reviews some of the most notable techniques which significantly improved the performances of the Deep Learning models. These include: the various gradient descent optimisers, dropout regularisation, batch Normalisation for Deep Learning models.

Advanced Gradient Descent Optimisers

Over recent years, various improvements have been proposed to the gradient descent algorithm which has greatly improved the performances of Deep Learning by accelerating learning. These gradient descent improvements include the Adagrad, Adadelta, RMSprop and ADAM. For a detailed mathematical overview of the various gradient descent alternatives mentioned see Appendix A. A brief summary of these alternatives to the basic gradient descent is given below.

Adagrad, [25] is an alternative gradient decent optimiser which has learning rates for each individual parameters and an adaptive learning rate which are calculated by dividing the learning rate by the sum of squares of the gradients. Adadelta [101] improves upon the Adagrad's adaptive learning rate by using a decaying average of the all past squared gradients. RMSprop and ADAM [49] are both advanced variants of the Adadelta. Notably, the RMSprop and ADAM optimisers are especially popular choices for training Deep Learning models.

Dropout Regularisation

Dropout is a stochastic regularisation method [82] which involves dropping out (zeroing out) a fraction of randomly selected units and their corresponding activations during the training time

by a certain probability of p. This technique was developed to combat overfitting of models caused by a phenomenon known as "co-adaptation" that occurs during training with can degrade generalisation performance. Dropout can be interpreted as a form of model averaging, which has the effect of approximately combining 2^n for n units i.e., exponentially many "thinned" neural network architectures formed from the different architectures that results from the dropout at training time. During training the dropout can be described modifying the activations of the network as follows:

$$r_i^{(l)} \sim Bernoulli(p), \tag{2.28}$$

$$\tilde{z}^{(l)} = r^{(l)} \odot z^{(l)}, \tag{2.29}$$

$$a_i^{(l+1)} = w_i^{(l+1)} \tilde{z}^{(l)} + b_i^{(l+1)}, \tag{2.30}$$

$$z_i^{l+1} = \sigma(a_i^{(l+1)}) \tag{2.31}$$

Therefore, each unit is retained with a fixed probability p independent of other units. A common setting for is $p = 0.5$ which empirically seems to work well for a wide range networks and tasks [82]. At test time all the units are made always active. Additionally, to account for the fact that units were deactivated by a rate of p during training, the weights of each unit are multiplied by p at test time as the weights of the network are a scaled-down versions of the trained network weights. This is to ensure that for any hidden unit the expected output is the same as the output during test time. The network's output at test time could therefore be described as:

$$a_i^{(l+1)} = w_i^{(l+1)} \tilde{z}^{(l)} p + b_i^{(l+1)}, \tag{2.32}$$

$$z_i^{l+1} = \sigma(a_i^{(l+1)}) \tag{2.33}$$

Batch Normalisation

Algorithm 4 Batch Normalisation [47]

Input: Values of x over a mini-batch $\mathcal{B} = \{x_1...m\}$: Parameters to be learned: γ, β
Output: $y_i = BN_{\gamma\beta}(x_i)$

1:
$$\mu\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$
//mini-batch mean

2:
$$\sigma^2\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i\mathcal{B})^2$$
// mini-batch variance

3:
$$\hat{x}_i \leftarrow \frac{x_i - \mu\mathcal{B}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$
//normalize

4:
$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta(x_i)}$$
//scale and shift

Batch Normalisation [47] is a Deep Learning technique which accelerates the training process of Deep Learning models while alleviating some of the difficulties of training a deep neural network model by reducing the **Internal Covariate Shifts** in the Deep Learning models.

To describe what the Internal Covariate Shifts is, it is worth considering that when training models that are especially deep inputs to each layer are affected by the parameters of all preceding layers. This would mean that small changes to the network parameters are amplified as the network becomes deeper. These changes in the distributions of each layer's inputs presents a problem because during training, the layers need to continuously adapt to the new distributions by re-adjusting and compensating for the changes in the distributions of their inputs. The *change in*

the distributions of internal nodes of a deep network during training are the Internal Covariate Shifts of the network and thus, eliminating it would in principle allow for a faster training [47].

The Batch normalisation aims to reduce the Internal Covariate Shifts by normalizing each layer's inputs based on its mini-batch statistics, to have a zero mean and unit variance and additionally, scaling and shifting them with two learnable parameters γ and β. The batch Normalisation algorithm is described in Algorithm 4.

When used in practice, batch normalized networks are able to realise gains of successful training with higher learning rates and tolerance to less desirable configuration of initialization weights. The original work which introduced the batch normalisation technique was able to successfully match the performance of the same model which did not use batch normalisation by using only 7% of the training steps and with further training, was able to exceed its accuracy by a substantial margin [47].

2.9 Deep Learning for Fine-grained Image Prediction

The popular Convolutional Neural Network described thus far, were used for image classification, which involves assigning a single category label to each image. Tasks of those kind only require the Deep Learning network to be sensitive to coarse category-level semantic information of the images.

There have been more recent Convolutional Neural Networks models which has been applied to more finer grained tasks such as image segmentation, object detection and parts labelling etc. which requires predictions to be made at pixel level. These are referred to as fine-grained tasks since they differ from image classification because in addition to the need to be sensitive to coarse information, the output requires localization i.e. labels need to be assigned for every pixel. Therefore these tasks are structurally different from image classification as they require *fine inference*.

These requirements have subsequently led to some proposal of architectural changes to the stan-

dard deep leaning models to be made, [60, 73, 34, 6]. These various improvements to the Deep Learning models for fine grained tasks include some of the following: replacing pooling layers with up-sampling layers by **Fully Convolutional Networks** [60]; allowing for the output layer to directly use features from previous layers via short-cut connections thereby allowing the coarse higher layer information to be combined with fine lower layer information as implemented by **Hypercolumn Networks** [34]; using an architecture which consists of contracting layers to capture context and a symmetric expanding layers that enable localization by **U-Nets** [73]. These modifications allow the networks to capture semantics with better localization.

2.9.1 Convolution Layers for Fine-grained Predictions

What many of the above fine-grained prediction models have in common are their convolution layers which are modified to perform well for pixel level prediction tasks. These modified convolution layers include the transposed convolution layers and dilated convolution Layers which are described below.

Transpose Convolution (Up-sampling)

The standard convolution layers described in previous sections were used to spatially down sample the input images. However, fine-grained task involves predictions of every pixel of the input image, thus requiring some up-sampling operations in order for the spacial dimensions of the output to match the sizes of the input images.

Traditionally, up-sampling operations were implemented by applying some interpolation rule[10] to rescale the output predictions, yet it would be desirable to directly learn the up-sampling transformations within the network. Modern architectures are able learn the up-sampling transformations through the Transpose convolution layers [102]. Transpose convolution layers (also variously referred to as fractionally strided convolution, up-convolution or deconvolution[11]) works by swapping

[10] Various interpolation rules exists such as the Nearest-neighbour interpolation and Bilinear interpolation etc.
[11] Although the term deconvolution often discouraged as the operation described here is different from the math-

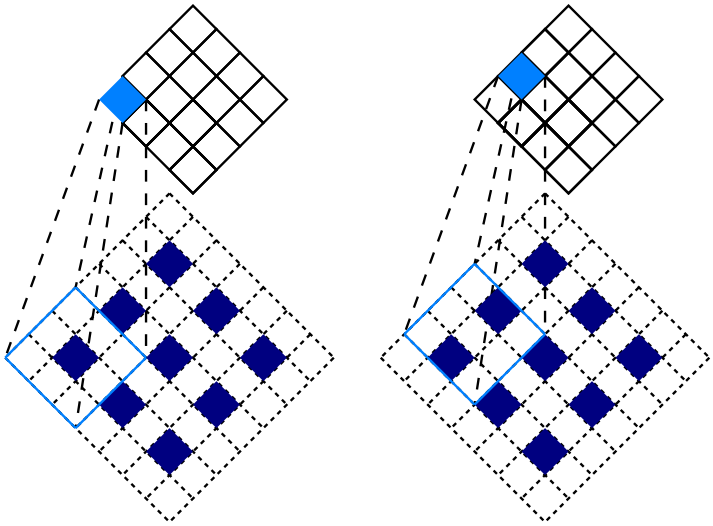


Figure 2.6. Transpose Convolution with direct Convolutions and paddings

the forward and backward passes of a convolution [26]. Additionally, it is also possible to implement an equivalent transposed convolution with a direct convolution, which involves adding rows and columns of zeros to the input (similar to padding) as shown in Figure 2.6, although this results in a much less efficient implementation.

Dilated Convolution

Another problem faced by fine-grained predictions is the problem of modelling long-range dependencies. A model's ability to learn long-range dependencies are affected by the effective receptive fields of the convolution layers. The effective receptive fields the area of the original image that can influence the activations outputs. Therefore for fine-grained predictions, large effective receptive fields which extends across large areas of the input image are desirable. With standard convolutions, the rate with which the effective receptive field expand is linear with each proceeding layer. Dilated Convolution layers [99, 12] are able to expend the effective receptive field exponentially while keeping the growth of parameter numbers linear. With Dilated Convolution, spaces as inserted between the kernels of the filters as shown in Figure 2.7 so as to "inflate" them.

ematical definition of deconvolution i.e., the inverse of convolution.

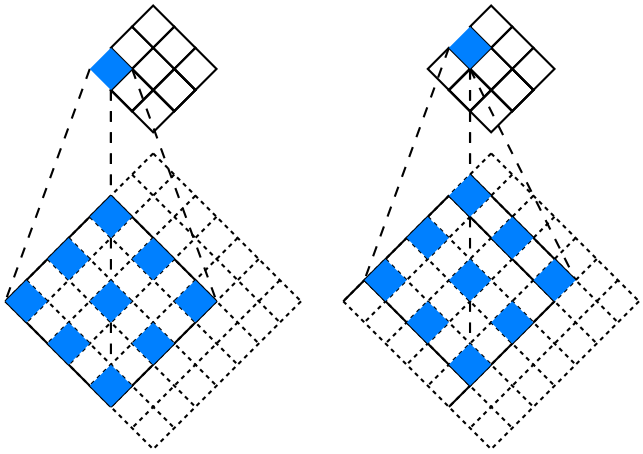


Figure 2.7. Dilated Convolution and the effect on the receptive field

The difference between the standard convolution and a dilated convolution is described as the following. Whereas a standard convolutions is:

$$(F * k)(p) = \sum_{s+t=p} F(s)K(t) \tag{2.34}$$

a dilated convolution is described as:

$$(F *_D k)(p) = \sum_{s+Dt=p} F(s)K(t) \tag{2.35}$$

The dilated convolution is controlled by a hyper-parameter D, where D - 1 spaces are inserted between the filter elements. Therefore, if $D = 1$, it is equivalent to a standard convolution. Figure 2.8 shows how the receptive fields in a dilated convolution expands with different values for D.

2.10 Unsupervised Deep Learning

The models described in previous sections involved supervised learning, i.e., learning with labels for the data supplied in advance. In unsupervised learning the purpose is to discover structures from

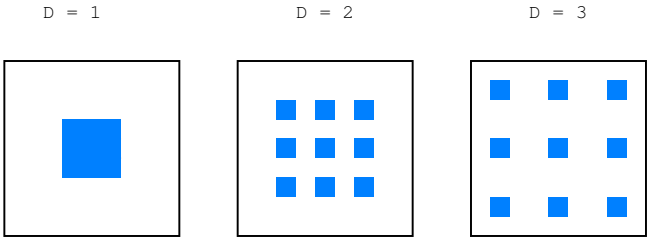


Figure 2.8. Dilated Convolution with different Dilation values

data and thus learning can take place even if the data with no labels are supplied. This section in particular, focuses on two kinds of Deep Learning models for unsupervised learning. First kind are the auto encoder models. The second is a generative model known as the generative adversarial networks.

2.10.1 Auto Encoders

Auto encoders [39] are deep neural network models that are used for unsupervised learning. The purpose of the auto encoder models are to learn meaningful representation of some given data by coping its inputs to its outputs with a **code** representation by using a bottle neck to force abstractions. Auto encoders have a two part structure which consists of **encoder** which maps the input into the code (a compressed representation of the inputs) and a **decoder** which maps the code back to the original input, thus reconstructing them from the code. The encoder ϕ and decoder ψ can therefore, be described as transitions such that:

$$\begin{aligned} \phi &= \mathcal{X} \rightarrow \mathcal{F} \\ \psi &= \mathcal{F} \rightarrow \mathcal{X} \\ \phi, \psi &= argmin_{\phi,\psi} \|X - (\phi, \psi)X\|^2 \end{aligned} \tag{2.36}$$

A simple auto encoder with one hidden layer where the input is $x \in \mathbb{R}^d = \mathcal{X}$ which maps to $\in \mathbb{R}^p = \mathcal{F}$ can be expressed as:

$$z = \sigma(Wx + b) \tag{2.37}$$

where W is the weight matrix, b is the bias and σ is an activation function. Using the decoder, the code z is reconstructed back to x' which has the same shape as x as follows:

$$x' = \sigma'(W'z + b') \tag{2.38}$$

the model is trained by minimising the reconstruction errors, which gives the loss function:

$$\mathcal{L}(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b')\|^2 \tag{2.39}$$

Auto encoders usually have dimensions that are lower than the dimension of the input features $\mathcal{X}$ which forces a bottle neck structure which compresses the representations being learnt. Deep auto encoders can be constructed by extending the depth of layers in the encoder and decoder layers.

Several variants of the auto encoders exist which includes: sparse auto encoders [56], denoising auto encoders [92], stacked denoising auto encoders [93] and variational auto encoders [50]. The sparse auto encoder uses the same ideas as the auto encoder while adding sparsity constraint to improve the representations being learnt. The denoisinng auto encoders learns representation by training a network to recover (de-noise) corrupted versions of the inputs, and the stacked denoising auto encoders extends this idea by stacking layers of denoising auto encoders. There also exists generative version of the auto encoder known as the Variational Auto Encoders [51]. More recently the Split-Brain auto encoder [105] was introduced which uses a special representation learning by performing the task of colourisation by using a network architecture with two sub-network paths to learn self supervised representations of images.

2.10.2 Generative Models

Generative Deep Learning models are a branch of unsupervised learning which attempts to describe how a dataset is generated taking a probabilistic approach. The generative models therefore includes a stochastic element which influences the individual samples generated by the model, this way they are able to generate novel examples that are outside of the learnt dataset. The generative models include the Variational Auto Encoders [51] and Generative Adversarial Networks. The following will focus on the the Generative Adversarial Networks [31].

2.10.3 Generative Adversarial Networks and Image Synthesis

The Generative adversarial network (GAN) is a generative Deep Learning approach which was introduced by Goodfellow *et al.* in 2014 [31]. GANs offer a framework for estimating generative models by training two neural networks which compete against each other in a zero sum game.

An adversarial network can be described as learning a distribution p_g over some data x using a Generator neural network $G(z;\theta_g)$ with parameters θ_g where z is noise generated from a Gaussian distribution p_z. Additionally, a second Discriminator neural network $D(x;\theta_d)$ is used to output a scalar value, which represents the probability that x came from the actual data rather than p_g (in other words a sample generated by G). D is trained to maximize the probability of correctly assigning labels to both examples from the data and samples generated by G, while G is also simultaneously trained to minimize $\log(1 - D(G(z)))$. Hence, D and G are playing a mini-max game with the value function $V(G, D)$:

$$\min_G \max_D V(D,G) : \mathbb{E}_{x\sim data_p(x)}(\log D(x)) + \mathbb{E}_{z\sim p_z(z)}(\log(1 - D(G(z)))) \tag{2.40}$$

Thus, GANs are trained by alternating between the two networks as follows:

- Gradient ascend for the Discriminator:

$$\max_{\theta_d}[\mathbb{E}_{x\sim data_p(x)}\log D_{\theta_d}(x) + \mathbb{E}_{z\sim p_z(z)}\log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \tag{2.41}$$

- and gradient descent for the Generator:

$$\min_{\theta_g}[\mathbb{E}_{z\sim p_z(z)}\log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \tag{2.42}$$

It should be noted that often in practice the loss of the Generator is implemented with a gradient ascent using $\log(D_{\theta_d}(G_{\theta_g}(z)))$ instead because of the difficulty of training the generator in the early stages of the training due to the loss landscape when using Equation 2.42. The complete algorithm for training a GAN could therefore be described in Algorithm 5 [31].

Algorithm 5 Generative Adversarial Network training [31]

for number of training iterations **do**
  for k steps **do**
    Sample mini-batch of m noise samples $\{z^1, ...z^m\}$ from noise prior $p_g(z)$
    Sample mini-batch of m examples $\{x^1, ...x^m\}$ from data distribution $p_{\text{data}(x)}$
    Update the Discriminator by ascending its gradient: $\nabla\theta_d \frac{1}{m}\sum_{i=1}^{m}[\log D_{\theta_d}(x^i) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^i)))]$
  end for
  Sample mini-batch of m noise samples $\{z^1, ...z^m\}$ from noise prior $p_g(z)$
  Update the Generator by ascending its gradient: $\nabla\theta_g \frac{1}{m}\sum_{i=1}^{m}[\log(D_{\theta_d}(G_{\theta_g}(z^i)))]$
end for

For the task of image synthesis, the Generator network is trained to generate images which look convincingly similar to the examples of real images in the dataset in attempts to 'fool' the Discriminator network. Thus when a GAN is trained successfully, it is able to generate novel synthesised images. It should be noted that GANs are difficult to successfully train because:

- There is a lack of heuristic loss function (a common choice for the image generator GANs loss is pixel-wise mean squared error).
- Training is difficult and unstable often resulting in non-convergence when the models never

converge which results in oscillating models or mode collapse when the outputs of the generator models produces limited varieties of samples.

Seeking improvements for GANs is therefore an ongoing and active area of research [79]. Among them GANs which uses Convolutional Neural Network was first proposed in 2015, known as the Deep convolution GAN (DCGAN) [71]. Also notably in 2016, Isola *et al.* proposed a conditional version of the GANs [48] as a generalised solution to image-to-image translations. Their approach [48] is able to perform a wide variety of image translation tasks including the colourisation of black-and-white images. More recently, GANs which uses self-attentions [103] have also been introduced.

2.11 Attention Mechanisms for Deep Learning

Attention is the cognitive process of selectively concentrating on discrete aspects of information, and is extensively studied in fields such as cognitive psychology. A recent trend in Deep Learning is to use various forms of attention to improve models. The attention in the context of its use in Deep Learning, is the use of a **vector of importance weights** as a means of biasing the allocation of available computational resources towards the most informative components of an input signal. Below gives a brief summary of the most impactful works concerning the use of attention mechanism for Deep Learning.

2.11.1 Attention with Deep Learning Models

Some of the earliest and most popular use of the attention models were for modelling languages. Specifically, an attention model by Bahdanau *et al.*, in 2014 [7] was proposed as an improvement for the Recurrent neural network[12] approach for the task of neural machine translation[13] [84, 15],

[12]These are neural networks architectures with cyclic connections, i.e. feedback loops which gives them implicit memory.

[13]Briefly, this approach involved an encoder-decoder architecture which encodes a source sentence into a fixed-length vector from which a decoder generates a translation of that sentence.

which suffered from a critical disadvantage of using a fixed-length context vector design which impeded the ability of the recurrent network to remember long sequences as evidenced by the rapidly deteriorating performances of the models as the length of the input sentences increased [15]. The attention mechanism by Bahdanau *et al.*, [7] alleviated some of the above problem by using an attention mechanism to jointly learn to align and translate sentences which allowed models to cope better with long sentences and improve the quality of the translations.

Another noteworthy use of the attention mechanism is the Neural Turing Machine (2014) developed by Graves *et al.*, [33]. Their model used an architecture which coupled an external memory with a neural network which used an attention mechanism to control operation heads which read or wrote to the external memory, mimicking the Turing machine's tape.

Attention models have also been used for the task of image captioning, where notably the work "Show, Attend and Tell" (2015) [97] by Xu *et al.*, used a conolutional neural network encoder to extract visual features which were fed to a recurrent neural network decoder to generate words with descriptive sentences of a given input images. Importantly, this work used a Stochastic 'Hard' and Deterministic 'Soft' attention models which learned to attend to salient parts of the input image while generating its caption, improving the captioning quality.

More recently, the work "Attention Is All You Need" (2017) by Vaswani *et al.*, [91] proposed an attention model based on a feed-forward neural network with attention mechanisms for machine translation, dispensing the need for recurrent architectures which were previously thought to be the best approach for language modelling, but are costly to train due to their un-parallelizable, sequential nature. Their model, called the transformer set a new state-of-the-art for a machine translation task while using only a fraction of the training costs of the best models in the literature at the time they were introduced. Additionally, an even larger-scale implementation of the transformer called the GPT-2 was introduced by Radford *et al.*, [72] which went on to achieve state-of-the-art results on 7 out of 8 language modelling benchmarking datasets.

2.11.2 Self-attention models for Convolutional Neural Networks

The following provides a more detailed description for two attention models in particular, which are featured in the subsequent chapters in this thesis. The two models are the Squeeze-and-Excitation Networks and the Self-Attention Generative Adversarial Networks.

Squeeze-and-Excitation Networks (2017)

The Squeeze-and-Excitation (SE) Network introduced by Hu *et al.*, [44] are Convolutional Neural Networks which uses a self-attention mechanism. This work achieved first place in ILSVRC in 2017 for the classification challenge using a Rsenet variant with self-attention modules. Their results also improved the best performance from the previous year by ∼25% (in relative improvement). The proposed self-attention module called the "Squeeze-and-Excitation" (SE) block enhances the performance of the networks by explicitly modelling channel interdependencies so that the network is able to increase its sensitivity to informative features [44].

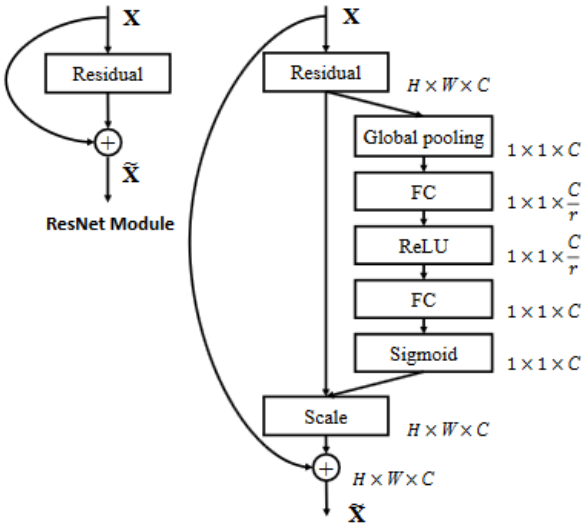


Figure 2.9. Squeeze-and-Excitation (SE) block in comparison to a Res block (Image source [44])

Figure 2.9 shows the structure for the SE modules in comparison to the standard Residual modules. The SE modules performs the convolution transformations with an additional two step process of:

- *Squeeze* which aggregates global spatial information into channel descriptors, using a Global average pooling layer.
- *Excite* which captures the channel-wise dependencies by using the information from the channel descriptors to re-calibrate the filter responses before they are fed into the next layer.

The SE block structure is shown in 2.9, (with a comparison to a standard Resdual block). A more detailed description of it can be given as follows. The SE block receives input $X \in \mathbb{R}^{H' \times W' \times C'}$ which is transformed to the feature maps $U \in \mathbb{R}^{H \times W \times C}$ via the standard convolution layers where $U = [u_1, u_2..., u_C]$. Then, the operation $\mathbf{F}_{sq}$ is used to generate a channel-wise statistics of U, which results in a vector $\mathbf{z} \in \mathbb{R}^C$ by shrinking U through its spatial dimension $H \times W$ such that the cth element of z is calculated by:

$$\mathbf{z}_c = \mathbf{F}_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{i=1}^{W} u_c(i, j) \tag{2.43}$$

In other words the above performs a Global average pooling which summarises the feature maps into a vector $\mathbf{z}$. The information aggregated in $\mathbf{z}$ is then fed to a gating mechanism $\mathbf{F}_{ex}$ which consists of a 2 fully-connected layers with non-linearities in between as follows:

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, W) = sigmoid(W_2 relu(W_1 \mathbf{z})) \tag{2.44}$$

where $W_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ and $W_2 \in \mathbb{R}^{C \times \frac{C}{r}}$ are the weights for the fully-connected layers which uses the hyper-parameter r to reduce the number of units in the hidden layers, hence it is a dimensionality-reduction ratio. The $\mathbf{s}$ can be though of an attention that provides a vector of feature importance. Finally, the output of the SE block is obtained by rescaling the feature maps U with the activations $\mathbf{s}$:

$$\tilde{x}_c = \mathbf{F}_{scale}(u_c, s_c) = s_c u_c \tag{2.45}$$

where $\tilde{X} = [\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_c]$ and $\mathbf{F}_{scale}$ refers to the channel-wise multiplication between the scalar s_c and the feature map $u_c \in \mathbb{R}^{H \times W}$.

Self-Attention Generative Adversarial Networks (2018)

The Self-Attention Generative Adversarial Network (SAGAN) [103] is an attention-based GAN which was recently introduced in 2018. The SAGAN is a convolutional GAN for image generation, which were able to improve on the quality of the images generated by introducing a self-attention mechanism which allows for efficient long-range dependency modelling. The SAGAN's attention mechanism is a module which are able to successfully enable both the generator and the discriminator networks to model relationships between widely separated spatial regions which helps in generating images with more complex geometry. In addition to the SAGAN also implements spectral normalisation to the weights which controls the training dynamics and two time-scale update (TTU) rule which varies the learning rate for discriminator and generator as a way to further stabilise the GAN training.

2.12 Mathematics for Colours

The purpose of this section is to familiarise the reader with colour concepts and colour spaces, and their mathematical relations. These will be most relevant to later chapters of this study which involves many of the transformation of colour spaces.

2.12.1 Basic concepts and terminology

As humans, colour is experienced as sensations when light in the visible region is reflected from a surface and enters the eyes through the pupil which is then focused onto the retina. The human retina normally have three types of colour photoreceptor cone cells which allow the perception of colour. These three types are the short-wavelength, medium-wavelength and long-wavelength cone which differ in their absorption spectra. By not taking into account the influences of the

spatial and temporal affect which influence perception, the sensation of colour is determined by the the three types of photoreceptor. Therefore, in theory only three numerical components are necessary and sufficient to describe a colour.

The **CIE**[14] has defined a system which specifies colours according to the human visual system known as the CIE system. The CIE system weights the spectral power distribution in terms of three colour matching functions, which are the sensitivities of a standard observer to a light. This weighting is done over the visual spectrum, 360nm to 830nm in set intervals. This requires a carefully defined lighting and viewing geometry as they affect the appearance of the colour and produces the **XYZ CIE tristimulus** values from which many colour measurements are made.

Colour spaces, (or colour models) are abstract mathematical model which describes the range of colours as tuples of numbers, which usually have three values. A colour space can be **device dependent or independent**. A colour space for which the resultant colour does not depend on equipment and set up are said to be device independent. CIELAB is an example of a device independent colour space. A colour **gamut** is the area enclosed by a colour space in three dimensions, and the range of colours available in the device independent colour space.

2.13 Colour Spaces and Transformation Formulae

This section introduces the RGB colour spaces and a few alternative colour spaces with their transformation formulae from RGB. Furthermore, the colour spaces below are limited to abstract colour spaces, which do not include colour spaces used for physical printing such as CMY (Cyan, Magenta and Yellow).

[14]CIE the (International Commission on Illumination) cie.co.at

2.13.1 RGB

RGB is a device dependant colour space and is one of the most common colour space for digital images as colour images are usually digitized in the RGB. The RGB colour space can be considered a cube space based on the three axis which correspond to red, green and blue. RGB are usually in 24-bits (8-bits for each R, G and B channel).

2.13.2 CIE XYZ

The CIE XYZ is a visual primary space based on the imaginary primary colour XYZ which have been selected so the all the colours can be which are perceived by the human eye are within the colour space. The XYZ system is based on the response curves of the three colour receptors of the eye which differ slightly from one person to person. The CIE has therefore defined a standard observer which corresponds to the average response of the population. There are various mathematical models to transform the RGB device dependant colour to the XYZ trsitmilus values. One of the simplest form of the model, is a simple matrix transform given below [17]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 3.06322 & -1.39333 & -0.475802 \\ -0.969243 & 1.87597 & 0.0415551 \\ 0.0678713 & -0.228834 & 1.06925 \end{bmatrix}^{-1} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.46}$$

2.13.3 CIELAB and CIELUV

CIELAB and CIELUV (or CIE $L^*a^*b^*$ and CIE $L^*u^*v^*$) are perceptually uniform colour spaces and are based on the CIE XYZ. They are a non-linear transformations which require a reference white (Z_0, Y_0, Z_0). They are an attempt to linearise the perceptibility of a unit colour difference, and are intended to mimic the logarithmic response of the eye. The transformation for the CIELAB and CIELUV is given as follows [19].

$$\begin{cases} L^* = 116(\frac{Y}{Y_0})^{\frac{1}{3}} - 16 & \text{if } \frac{Y}{Y_0} > 0.008856 \\ L^* = 903.3(\frac{Y}{Y_0}) & \text{if } \frac{Y}{Y_0} \leq 0.008856 \\ a^* = 500[\, f\frac{X}{X_0} - f\frac{Y}{Y_0}] \\ b^* = 200[\, f\frac{Y}{Y_0} - f\frac{Z}{Z_0}] \end{cases} \tag{2.47}$$

where

$$\begin{cases} f(U) = U^{\frac{1}{3}} & \text{if } U > 0.008856 \\ f(U) = 7.787U + 16/116 & \text{if } U \leq 0.00885 \end{cases} \tag{2.48}$$

and

$$\begin{cases} U(X,Y,Z) = \frac{4X}{X+15Y+3Z} \\ V(X,Y,Z)\frac{9Y}{X+15Y+3Z} \end{cases} \tag{2.49}$$

$$\begin{cases} L^* = 116(\frac{Y}{Y_0})^{\frac{1}{3}} - 16 & \text{if } \frac{Y}{Y_0} > 0.008856 \\ L^* = 903.3(\frac{Y}{Y_0}) & \text{if } \frac{Y}{Y_0} \leq 0.008856 \\ u^* = 13L^*[\, U(X,Y,Z) - U(X_0,Y_0,Z_0)] \\ v^* = 13L^*[\, V(X,Y,Z) - V(X_0,Y_0,Z_0)] \end{cases} \tag{2.50}$$

2.13.4 YUV and YIQ

The YUV and YIQ are colour spaces for video standards which were commonly used for analogue television transmission, they are therefore often called transmission primaries. The transformations for these are as follow[19]:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 \times R + 0.587 \times G + 0.114 \times B \\ -0.147 \times R - 0.289 \times G + 0.436 \times B \\ 0.615 \times R - 0.515 \times G - 0.100 \times B \end{bmatrix} \tag{2.51}$$

and

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 \times R + 0.587 \times G + 0.114 \times B \\ 0.596 \times R - 0.274 \times G - 0.322 \times B \\ 0.212 \times R - 0.523 \times G + 0.311 \times B \end{bmatrix} \tag{2.52}$$

2.13.5 HSV and HSI

The HSV and HSI are alternative colour spaces which were developed as a more intuitive colour space for human observers. In particular the HSV colour space which was invented by Alvy Ray Smith stands for Hue, Saturation and Value. The HSI stands for Hue, Saturation and Intensity. For HSV the transformation is given below as follows[15]:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} R/255 \\ G/255 \\ B/255 \end{bmatrix} \tag{2.53}$$

$$Cmax = max(R', G', B'), Cmin = min(R', G', B') \tag{2.54}$$

$$\triangle = CmaxCmin \tag{2.55}$$

$$H = \begin{cases} 0^\circ, & \triangle = 0 \\ 60^\circ(\frac{G'-B'}{\triangle} mod 6), & Cmax = R' \\ 60^\circ(\frac{B'-R'}{\triangle} + 2), & Cmax = G' \\ 60^\circ(\frac{R'-G'}{\triangle} + 4), & Cmax = B' \end{cases} \tag{2.56}$$

[15]https://www.vocal.com/video/rgb-and-hsvhsihsl-color-space-conversion/

$$S = \begin{cases} 0, & Cmax = 0 \\ \frac{\triangle}{Cmax}, & Cmax \neq 0 \end{cases} \tag{2.57}$$

$$V = Cmax \tag{2.58}$$

For HSI the transformation is as follows [19]:

$$\begin{cases} H & = arctan(\frac{\beta}{\alpha}) \\ S & = \sqrt{\alpha^2 + \beta^2} \\ I & = (R + G + B)/3 \end{cases} \tag{2.59}$$

where

$$\begin{cases} \alpha = R - \frac{1}{2}(G + B) \\ \beta = \frac{\sqrt{3}}{2}(G - B) \end{cases} \tag{2.60}$$

The purpose of this section was to provide the reader with a few colour concepts. In particular, the above provides some information on colour spaces which will become most relevant in later chapters of this study which involves transformations of colour spaces. In the above section, the details for several colour space transformations from the RGB were given. It is interesting to observe many of the intricate transformations of colour spaces. When transforming colour spaces from RGB, though not all, many of them are non-linear, with some even having cyclical relations.

Chapter 3

Self Supervised Deep Learning with Colour: Image Colourisation vs Contrastive Learning

3.1 Introduction

At present, supervised learning is the most predominant approach for training Deep Learning models, which involve using large databases containing hand labelled data which are used as supervisory signals. An alternative to the supervised learning are known as the self supervised learning methods which require no hand labelled data. This chapter compares two kinds of self supervised learning. The first kind are those which attempt to solve a predictive, hand-crafted pretext tasks by either systematically or stochastically corrupting the inputs and attempting to recover them, thus requiring no manual labels. An example of this method include using a Deep Learning model to solve image colourisation by corrupting input images systematically by separating them by L and ab channels [106]. The other kind are known as the contrastive learning methods which learn representations by contrasting data points of positive and negative examples, where positive examples are data points that are congruent to each other, while negative examples are data points which are incongruent to each other.

It was hypothesised that colourisation would seem to serve as a reasonable task for learning visual representations because predicting colours require object level reasoning, and thus, a model which can solve colourisation may have useful representations which could be re-used i.e., transferred to other vision tasks. Yet in practice, these representations are found to lack the generality to solve other more basic tasks such as image classification. contrastive learning methods methods on the

other and have been shown to be more success in learning representations that can generalise to other vision tasks.

This motivates an investigation for a self supervised learning method which can combine the task of predicting colours with contrastive learning framework, and if there are any benefits from this method.

3.2 Overview

The remainder of this chapter is structured as follows: Section 3.3 begins by providing some context around the works that are related to this chapter. Section 3.4 investigates the how useful the predictive task of image colourisation is as a self supervised learning method for visual representations by reviewing a few existing methods and conducting a short experiment to test the transferability of the learnt features using one of the method. Section 3.5 compares predictive and contrastive learning self supervised methods and proposes a method to combine the two, for which an experiment is conducted to evaluate the proposed method against previous methods. Section 3.6 provides a summary for this chapter, some direction of future works as concluding remarks.

3.3 Related Work and Context

This section aims to provide some context around the related works. There are various forms of self supervised learning which relies on solving some predictive pretext tasks in which the inputs and labels can be both derived from the raw image data. These are considered the predictive self supervised learning. Some examples of the predictive tasks include: image colourisation [104, 106, 55], solving jigsaw puzzles [67], predicting rotation angle of rotated images [28] and predicting relative positions of a patch for a corrupted image [24]. Proponents of these predictive form of self supervised learning would argue that representations learned through solving many of the above tasks are sufficiently generalisable to other tasks in the vision domain making them

useful as self supervised tasks.

A yet another kind of self supervised learning are known as the contrastive learning methods which have recently become increasingly popular as they been shown to learn better visual representations than the predictive self supervised methods, these often involve contrasting pairs of examples from the same data by creating different views from them. Some examples of these include: learning by classifying whether a pair of global features and local features are from the same image [40]; learning representations from contrasting sequential data such as speech, images, text and reinforcement learning - Contrastive Predictive Coding (CPC)[69];, learning representations from contrasting different views of the same image Contrastive Multiview Coding (CMC)[88]; and contrastive learning based on two differently augmented views of the same image (SimCLR) [13].

3.4 Colourisation as a Predictive Tasks for Visual Representation Learning

This section begins by first investigating if the predictive task of image colourisation alone is useful for Convolutional Neural Network models to learn useful visual representations. To this end, the following introduces two methods for self supervised feature learning using image colourisation, known as the Cross-channel auto encoders [104] and the Split-brain auto encoders [106]. For the latter of the two methods above, a short experiment is conducted with the aim of testing the transferability of the learnt features from the method for the down-stream task of image classification.

3.4.1 Cross-Channel Encoders

The Cross-channel encoders is a self supervised learning method which involves learning a representation using a convolutional network for inputs $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ ie., the images with with H height W width and C channels. The input is split into data $\mathbf{X_1} \in \mathbb{R}^{H \times W \times C_1}$ and $\mathbf{X_2} \in \mathbb{R}^{H \times W \times C_2}$ where $C_1, C_2 \subset C$. These subsets can be a split between the L and ab channels of the image for the Lab colour spaces. The Convolutional Neural Network is then tasked to solve $\hat{\mathbf{X}_2} = \mathcal{F}(\mathbf{X_1})$. The

function $\mathcal{F}$ is learnt by a convolutional neural network which produces a layered representation of input $\mathbf{X}_1$. By performing the colourisation task of predicting $\mathbf{X}_2$ from $\mathbf{X}_1$, the representation $\mathcal{F}(\mathbf{X}_1)$ is learnt. The network can be trained on various loss such as the regression $l2$:

$$l_{l2} = \mathcal{F}(\mathbf{X}_1, \mathbf{X}_2) = \frac{1}{2} \sum_{h,w} \|\mathbf{X}_{2,h,,w} - \mathcal{F}(\mathbf{X}_1)_h, w\|_2^2 \tag{3.1}$$

Or alternatively a classification loss can be used by encoding the output $\mathbf{X_2} \in \mathbb{R}^{H \times W \times C_2}$ into a distribution $\mathbf{Y_2} \in \triangle^{H \times W \times Q}$ using a function $\mathcal{H}$ where Q is number of elements of the output space which was quantized. In this case the Convoluational Network is trained to predict the distribution $\hat{\mathbf{Y}}_\mathbf{2} = \mathcal{F}(\mathbf{X_1} \in \triangle^{H \times W \times Q})$ where a cross entropy loss can be used as follows:

$$l_{cl} = (\mathcal{F}(\mathbf{X_1}), \mathbf{X_2}) = -\sum_{h,w} \sum_{q} \mathcal{H}(\mathbf{X_2})_{h,w,q} log(\mathcal{F}(\mathbf{X_1})_{h,w,q}) \tag{3.2}$$

3.4.2 Split-Brain Auto Encoders

However, the disadvantage of the Cross-channel auto encoders is that as the prediction is from one set of channels to the another, therefore at inference time one set of channel is lost (for example by learning a mapping $L \rightarrow ab$, the signals from channel ab is lost since they are discarded due to the input nodes of the architecture). The Split-Brain auto encoder [106] was an extension to solve this problem by making use of the entire input signal by training two dis-joint, sub-networks, which concatenate features. Split-Brain auto encoder consists of two sub-network, which each predicts one subset of channels of the input from the other such that the sub-networks $\mathcal{F}_1$ and $\mathcal{F}_2$ are optimised to minimise the loss functions L_1 and L_2 as follows:

$$\mathcal{F}_1 = argmin_{\mathcal{F}_1} L_1(\mathcal{F}_1(\mathbf{X_1}), \mathbf{X_2}) \tag{3.3}$$

$$\mathcal{F}_2 = argmin_{\mathcal{F}_2} L_2(\mathcal{F}_2(\mathbf{X_2}), \mathbf{X_1}) \tag{3.4}$$

where the loss functions L_1 and L_2 can be either the classification or regression losses l_{l2} or l_{cl} described previously. In the most simplest case, the Split-Brain auto encoder is used to solve the colourisation problem in Lab space, i.e. one sub-network learns to predict the L channel from the ab channels, while the other learns to predict the ab channels from the L channel (it is also possible extended this to other color spaces and splits using other kinds of channels beyond colours such as RGB and the depth in RGB-D images). The learnt representations are then concatenated layer wise at each layer l as $\mathcal{F}^l = [\mathcal{F}_1^l, \mathcal{F}_2^l]$. This way, the input and the output to the network $\mathcal{F}$ is the full input X. The original implementation of the Split-brain auto encoder consisted using a base model of AlexNet by splitting the architecture in the middle i.e., dividing the number of features in each layer by two to create two dis-joint sub-networks which are concatenated layer wise, although in principal any convolutional network architecture can be used to implement the Split-brain auto encoder.

3.4.3 Experiment

The purpose of this experiment is to investigate the Split-Brain auto encoder's ability to transfer the learnt features from image colourisation to a classification task. The experiment involves two phases 1) implementing the self supervised learning though the pretext colourisation task, and 2) fine-tuning the network with the learnt features to be used for the classification task. For the experiment that follows, several implementations of the Split-brain auto encoders with slight variations in the fine-tuning phase are used, which are compared against a Fully-supervised baseline model trained on the same quantity of labelled data. The experiment is conducted on the challenging unsupervised learning benchmarking dataset - STL10 [18] (an overview and samples of images can be found in Section: 1.4). In the following implementations, the ResNet18 architecture is adapted as the base model for the Split-brain auto encoder which offers the simplicity of implementation, robust performance, and due to their popularly used among comparative studies.

Additionally, the choice of using the most light weight variant of the ResNet, i.e., the ResNet18 (instead of ResNet50 or ResNet101 etc.) was simply a matter of feasibility due to limited computation resources.

Self supervised Learning with ResNet Split-Brain Auto Encoders

For the Split-Brain auto encoder implementation all images in the STL10 datasets are first converted into Lab colour space to be used as inputs. The Split-Brain auto encoders architecture was created by the splitting a ResNet18 architecture to form two dis-joint sub-networks as described in Section 3.4.2. The self supervised Learning for Split-Brain auto encoders receive input image signals for the colour channels of either L or ab for each sub-network. The networks are optimised to minimise the loss between the predicted and input colour images for the L to ab, or ab to L mapping at pixel level for each sub-network. The colour predictions are made at a lower resolution than the original image. The loss functions was varied so that either a classification (cross-entropy) loss is used by quantising the L and ab spaces (into 100 and 256 bins for L and ab channels respectively), or by using a regression (L2) loss for the L and ab values between the predictions and the ground truth colour images. All self supervised training for the Split-Brain auto encoders used the Unlabelled set of the STL10 which include 100,000 images was optimised using the Adam optimiser with learning rate 1e-3 for 5 epochs.

Additionally, to improve the quality of the image colourisation during the self supervised learning phase, the Split-Brain auto encoder models which uses the classification loss functions are trained using a tailored classification loss function to re-balance the penalty of the classes, (more details for the loss function tailored to image colourisation provided in Section 4.4.1). This was done to combat the negative effects of the severe imbalance found in the statistical distribution for the quantised L and ab pixel values for natural images which may hinder the optimisation (among them is the risk of the models simply learning to predict a majority class colour in the quantised L and ab channels). To present some evidence of this imbalance, Figures 3.1 and 3.2 show the statistics on the pixel distributions of L and ab values for all the unlabelled STL10 images. In particular, histogram in Figure 3.1 shows the distribution of L channel pixels has a 'spike' at the

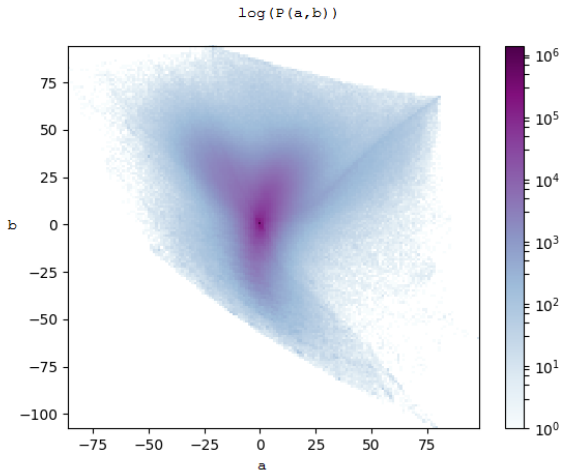


Figure 3.1. Distribution (in Log scale) of the a and b values for the unlabelled set in the STL10 dataset

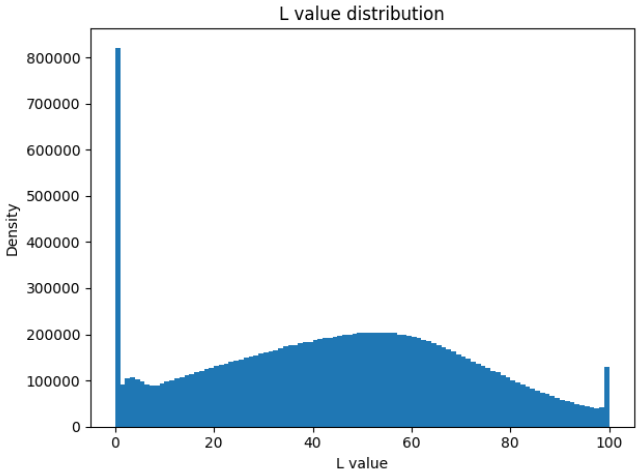


Figure 3.2. distribution of the L values for the unlabelled set in the STL10 dataset

0 and 100 values. Investigating further it was found that some of the images where sometimes cropped, leaving the cropped area black (0) while others are taken against a while background (100). Figure 3.2, show a concentration around 0 due to the ab values in natural photos having a tendency to be biased towards greyish desaturated colours [104].

Fine-tuning for Classification Task

For the Fine-tuning phase of the experiment, an MLP classification layer is added on top of the concentrated representations of the Split-Brain auto encoders models. The Split-Brain auto encoders are fine-tuned to perform the classification task using on the training set portion of the STL10 dataset which contain the labels for the images, by minimising the classification loss between the predicted class and the ground truth labels supplied by the STL10 dataset. Furthermore, when fine-tuning the models, two variants of fine-tuning are implemented described below. For both variants the models are trained for 400 epochs.

- **Freeze & Training Top-layers**: by fine-tuning only the top 2 layers of the network while freezing (fixing) the rest of the network weights. Only top-layers are fine-tuned using the

Adam optimiser with a learning rate of 1e-2.

- **Fine-tune with Differential Learning** [98]: where the whole network is fine-tuned, with different learning rates. The Fine-tuning with Differential Learning uses the adam optimiser with a learning rate of 1e-2 for the top 2 layers of the network and 1e-3 for the rest of the network.

Fully Supervised Models (Baseline)

In addition, the Split-Brain auto encoders described above are also compared against a ResNet18 model with RGB inputs, trained fully-supervised on the classification task, initialised with random weights using only the training set portion of the STL10 dataset. These models were optimised using the Adam optimiser with learning rate 1e-2, with dropout applied with the rate of 0.5 for 400 epochs to minimise the classification loss between the predicted class and the ground truth labels. These models serve as a baseline to the self supervised alternatives for the experiment.

Evaluation

The final results are given as the Classification accuracy score for the model performance of the STL10 test set unseen to the models which involves comparing the model's predicted class with the ground truth labels. The classification accuracy is the percentage of correct predictions made by the model. The evaluations are carried out in two ways: 1) by using only 1000 labelled examples from the training set to train the model according to 10 predefined folds supplied by the authors [18] for which the performance is averaged for all folds or 2) by using all 5000 training examples provided in the dataset to train a model.

3.4.4 Results

Method	1000 Labels	5000 Labels
Fully Supervised (Baseline)		
Random Initialisation + Dropout	53.59±0.7	69.63
Freeze body & Training Top-layers		
Re-Balanced Classification	51.87±0.56	64.24
Fine-tune with Differential Learning		
Re-Balanced Classification	54.78±0.58	68.7
Regression	53.66±0.64	69.07

Table 3.1. STL10 Classification accuracy performance of standard ResNet Split-Brain auto encoders

The results of the experiment is given in Table 3.1. The table shows the classification accuracy in percentage for the test set images in STL10 for each of the varying methods. For models trained with 5000 labelled images are available for training set, the self supervised learning methods do not offer any advantage over the supervised learning method as they do not obtain better performance. When only 1000 labelled image are available for training set, the self supervised learning methods obtains a gain of no more than 1 percent by when using a classification-based loss Split-brain auto encoders. When comparing between the different loss functions used for the self supervised learning phase, it is difficult to find a clear winner between the the classification and regression losses. The above results also show that when fine-tuning a model, the differential learning method is recommended, as shown by better classification results. This may be from the flexibility the model to adapt to the classification task when compared to freezing the weights.

3.4.5 Discussion

From the above results, the loss functions used (classification or regression) during the self supervised learning phase of Split-brain auto encoder models seem to have little effect on the quality

of the learnt features. The above results also show that the features learnt from the colourisation task may not be generalising well enough to the down-stream task of image colourisation when comparing the differential learning method with the fixing (freezing) weights method.

It has been previously hypothesised that image colourisation provides a reasonable pretext task for self supervised learning because predicting colours require object level reasoning, and thus, a model solving colourisation would have useful representations which could be re-used i.e., transferred to other vision tasks. However, the results presented in the experiments above however, do not provide encouraging evidence in support of this, but rather reveals the difficulty learning visual representations based on image colourisation alone. For the down-stream task of image classification the feature representations learnt from colourisation perhaps lack generality and transferability of the learnt features. These results motivates the next section which explores other alternative methods of self supervised learning for visual representations that can better generalise and uses colour information.

3.5 Visual Representation Learning with Colours and Contrastive Learning

This section compares the predictive and contrastive self supervised methods for visual feature learning, and in addition also investigates means to combine the two. The following will first introduce a simple contrastive learning method, and describe how a few contrastive learning methods can be applied in practice. After that, an alternative method which combines the contrastive learning methods with the predictive (colourisation) learning is proposed. Lastly, an experiment which evaluates and compares the representation quality learnt from the predictive, contrastive and the proposed alternative is carried out based on a linear evaluation method.

3.5.1 Learning From Augmented Views: Simple Contrastive Learning (SimCLR)

SimCLR [13] is a recently introduced contrastive learning algorithm which learns representations by maximizing the agreement between differently augmented views of the same data example

using a contrastive loss. The framework consists of four major components:

- A stochastic data augmentation module is used on a data example which results into correlated view of the same example denoted by $\tilde{x_i}$ and $\tilde{x_j}$ which is considered a positive pair.
- A Neural Network base encoder $f(\cdot)$ which extracts representations $h_i = f(\tilde{x_i})$. In the original implementation, a ResNet was used thus, $h_i = f(\tilde{x_i}) = \text{ResNet}(\tilde{x_i})$ where $h_i \in \mathbb{R}^d$
- A small Neural Network projection head $g(\cdot)$ maps the representation where the contrastive loss is applied, which uses a MLP with one hidden layer with a ReLU activation.
- A contrastive loss function defined for a contrastive prediction task. Where given a set $\{\tilde{x_k}\}$ which includes positive pair examples of $\tilde{x_i}$ and $\tilde{x_j}$, the contrastive prediction task aims to identify $\tilde{x_j}$ in $\{\tilde{x}_{k[k\neq i]}\}$ for a given $\tilde{x_i}$.

A mini batch of N examples are sampled for the contrastive prediction task on pairs of augmented examples derived from the minibatch, resulting in 2N data points. Given, $\text{sim}(u, v) = u^\top v / \parallel u \parallel \parallel v \parallel$, i.e., the cosine similarity between vectors u and v, the loss function for a positive pair of examples (i, j) is defined as

$$l_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}, \tag{3.5}$$

where $\mathbb{1}_{[k\neq i]} \in [0, 1]$ which is 1 iff $[k \neq i]$ and τ denotes a temperature parameter.

At the time of writing, the SimCLR is a state of the art among contrastive learning methods while presenting a simple approach to self supervised learning compared to other comparable methods. Among its achievement, the SimCLR is able to outperform the fully supervised AlexNet models with 10 times fewer labelled training data for classification task with the ImageNet benchmarking challenge.

3.5.2 Contrastive Learning with Images

The following describes two simple instances of contrastive learning with images:

Contrastive Leaning on differently augmented views of the same images. The SimCLR framework suggests creating pairs of differently augmented views of the same image by using a stochastic data augmentation modules which distorts the images by randomly applying some of the following to the images: 1) randomly cropping of the image, 2) applying colour distortion to the image (which involves randomly changing the brightens, contrast, saturation, or hue), grey-scaling (colour dropping), and 3) applying some Gaussian blur to the image.

Contrastive Leaning on different views of images. Alternatively, different views of the data can also be considered to create pairs (e.g., luminance and chrominance of the same image) as previously suggested by Tian *et al.,* [88]. For a simple case, using the *Lab* colour space would suffice, i.e., the L and ab channels from the same image can be considered as the positive pair.

3.5.3 Combining Contrastive Objectives with Colourisation

This section additionally investigates an alternative method for self supervised learning by proposing a method that combines the image colourisation (predictive) task with the contrastive learning method described above. This involves jointly solving the image colourisation task, contrastive learning task for differently augmented views of images, using a special network architecture described below.

The Proposed Network Architecture

The structure of the network architecture proposed is shown in Figure 3.3. It consists of a base encoder which comprises two-disjoint sub-networks as proposed by the Split-brain auto encoder, which splits the Lab channels to receive one of either the L or ab channels which contain different views of the image. Each of the sub-networks are tasked to solve the image colourisation task

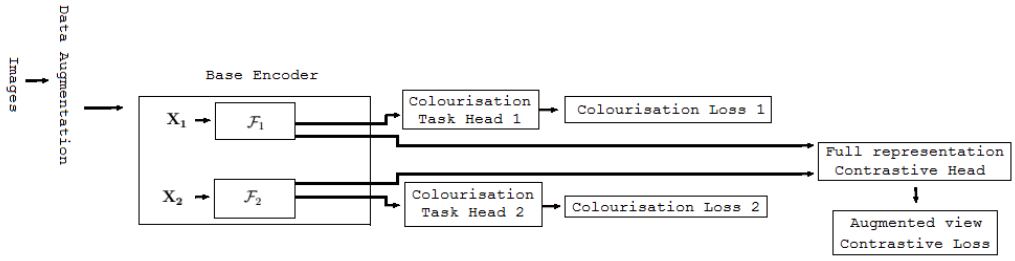


Figure 3.3. Structure of the Proposed Predictive/Contrastive Learning Network

using a dedicated task head. The colourisation task heads are convolution layers which predicts the pixel values of the colours for the image for the L or ab channel opposite to the channel which the sub-network received where the L2 regression loss is applied. Additionally, the sub-networks are concatenated layer wise for the full representation of the base encoder which has a projection head which is a MLP with a single hidden layer with ReLU non-linearity where the contrastive loss based on differently augmented views (i.e., the contrastive loss defined for the SimCLR) is applied. For each training iteration, the total loss of the base encoder is calculated as the sum of the contrastive loss and the colourisation losses from each of the sub-networks. The

3.5.4 Experiment

The aim of this experimental section is to compare the quality of the learnt representations for predictive (colourisation), contrastive and the proposed self supervised learning methods by learning representations from the unlabelled set of the STL10 dataset. The following provides details for the self supervised methods used to compare with the proposed method and the evaluation procedure used for the experiment.

Compared Self supervised Methods

To keep the comparison fair, the following self supervised methods described all adapt the Resnet18 architecture with a representation dimension of 512, as the base encoder or the equivalent Split-brain auto encoder by splitting the parameters of the Resnet18 architecture in half. All of the methods below are trained for 20 epochs with batch size 128, on the unlabelled potion of the STL10 dataset which contain 100,000 images.

- **Split-Brain auto encoder** The Split-Brain auto encoder is implemented as described in previous sections by training on the colour prediction task using L2 regression loss.
- **SimCLR** The SimCLR is implemented as described in Section 3.5.1 with a projection head which has 1024 dimensions and temperature parameter $\tau = 0.5$ using the data augmentation module described in Section 3.5.2.
- **Contrastive Learning on Channels L and ab** In addition to the SimCLR, a contrastive learning method based on the Lab channels is implemented which adapts the Split-brain auto encoder architecture as a base encoder and uses the same contrastive loss set-up as the SimCLR framework, except instead of using data augmentation to create congruent pairs, different views of the Lab channels from the images are used to create the pairs i.e., the L and ab channels from the same image is considered a positive pair.
- **The proposed method** The proposed method is impelmented as described in Section 3.5.3. It should be noted that the colour predictions are made despite the colour distortions being applied to the images from the data augmentation module, which makes the task more challenging.

Evaluation

The learnt representation are evaluated by freezing the weights of the base models and a training a (linear) logistic regression classifier[1] using the labelled data on top of the models for each methods from which the test set accuracy is used as a proxy for evaluating the representation quality. Using the STL10 dataset, allows for 2 evaluations for each method 1) by using only 1000 labelled examples from the training set to train the linear classifier according to 10 predefined folds for which the performance is averaged, and 2) by using all 5000 training examples provided in the dataset to train the linear classifier.

3.5.5 Results and Discussion

		Linear evaluation	
Method	Self supervised learning type	1000 Labels	5000 Labels
Split-Brain auto encoders	Predictive	32.8	39.2
Contrasting channels (L and ab)	Contrastive	35.9	41.3
SimCLR	Contrastive	43.1	49.7
Proposed method	Predictive/Contrastive	45.7	52.3

Table 3.2. Linear separability of different methods for the STL10 test set

Table 3.2 shows the results for the test set classification accuracy of the linear model calculated in percentage. The results shows that the contrastive learning methods are better for learning visual representations compared to the predictive method (i.e, the Split-brain auto encoder method which performed the worst) as shown by linear evaluation. Notably the proposed method showed a notable empirical gain over the already strong performance of SimCLR.

The results from the above experiment is promising particularly because it shows that when learning visual representations, the contrastive learning and predictive learning methods can be

[1]The linear classifier used was based on the scikit-learn machine learning library: software available from github.com/scikit-learn/scikit-learn.

applied jointly as a self supervised method, and can improve the quality of the learnt representation for Convolutional Neural Network models.

3.6 Conclusion and Future Work

This section concludes by providing a summary of this chapter and mentioning some future work. The aim of this chapter was to explore self supervised learning methods for learning visual representations and to explore what roles colours could play in achieving this. Previously, self supervised learning methods based on teaching a Deep Learning to solve various pretext tasks such as image colourisation has been proposed. This work therefore reviewed two of these methods known as the Cross-channel auto encoder [104] and the Split-brain auto encoders [106] However, through experiment it was found that on a practical level, the representations learned from the Split-brain auto encoders lacked the generality to be useful for solving classification tasks.

The above therefore motivated the later sections of this chapter to investigate more recent approaches for self supervised learning which involve contrasting pairs of examples from the data by creating different views from them i.e, Contrastive Leaning methods, and particular reviewed a simple framework for this known SimCLR [13]. An experiment that was conduced which compared the different self supervised learning methods found that the Contrastive Leaning methods are better at learning visual representation than the Split-brain auto encoder method. Moreover, a proposed self supervised learning method which performs both the image colourisation task jointly with contrastive learning was able to demonstrate notably improved results over the previous methods tested in the experiment.

The experiments in this chapter is limited by the fact that the visual representations are evaluated only for the downstream classification task, and therefore expanding variety of downstream task for evaluation is likely to benefit this study.

Chapter 4

Efficient Generative Adversarial Image Colourisation

4.1 Introduction

This chapter explores Deep Learning for the graphics task of image colourisation. Colourisation is the process of adding colours to black and white images which were done traditionally by artist who scrupulously hand-colour the images. As a computer graphics problem, image colourisation is highly ill-posed due to the multi-modal nature of the task since objects and scenes can appear in various plausible colours. Despite this, in recent years, there have been significant interest in automating the colourisation task by using Deep Learning models by mapping gray-scale inputs images to colourised output images.

Many Deep Learning systems that achieve visually impressive results often require extremely large Convolutional neural network models with excessive computation requirements, large memory footprints and lengthy training time to converge. Subsequently, previous Deep Learning approaches have also required either the need for a complex architecture which extract multiple levels of abstractions from the input images or the use of complex hand engineered loss functions. Despite this, the convolution layers used by conventional Deep Convolutional neural networks have been found to have trouble in generating complex geometries and modelling long-range spatial dependencies.

In particular this work focuses on network architecture design with efficiency of computational

resource in mind, in addition to building on important insights from state of the art Deep Learning models for image synthesis. By using depth wise separable convolution layers, this work shows that model architectures can be streamlined to have low memory requirements, and reduce computation costs. By adapting a Generative Adversarial Network to the colourisation problem, a structured loss is learnt through the process of training, allowing the network to be unconstrained by a complex hand-crafted one. The Self-attention mechanism for generative adversarial networks as proposed by Zhang *et al.,* [103] (originally applied for image synthesis models) are found to be better able to handle the modelling of long-range spatial dependencies when compared to conventional Convolutional layers further improving output quality.

4.2 Overview

The remainder of this chapter is structured as follows: Section: 4.3 gives a brief survey on the previous Deep Learning approaches to image colourisation. Section: 4.4 delves into more technical details of the colourisation problems and in particular discusses two common approaches to image colourisation using Deep Learning. Section: 4.5 provides details to the proposed method which is implemented in this work. Section: 4.6 provides a some qualitative results for the proposed method. Finally, Section: 4.7 provides a conclusion for this chapter.

4.3 Related Work

The following provides a brief survey on the previous Deep Learning approaches to image colourisation listed roughly in the order of appearance. It should be noted that the related works below only include systems which colourise images automatically and excludes systems which require any additional user inputs or interactions:

- Dahl's work [22] is an unpublished early colourisation model which uses ImageNet features on a pre-trained VGG model which was trained on a regression loss.

- Iizuka *et al.*, [46] introduced a colourisation model where a two-stream architecture is used which fuses global, medium and local features allowing the colourisation output to be based on many levels of abstractions of the input grey scale image.

- Zhang *et al.*, [104] proposed a VGG styled network with added depths of up-sampling and dilated convolutions and defined a classification based loss which rebalances the classification penalty for rarer colours to promote more diversity in the predictions.

- Larsson *et al.*, [55] used a VGG network with hyper-columns which allows features from higher level layers to be connected with layers deeper in the network. The model predicts a histogram distribution of colours.

- Pix2pix by Isola *et al.*, [48] generalised the conditional GANs as a general propose solution for image-to-image translation, solving among them the colourisation task. This approach involves an adversarial training where a Generator network minimises a loss objective which combines the l1 distance (between the real ground truth image and the generated image) and an adversarial loss, while the Discriminator network evaluates on patches of the real image and fake generated images.

- Nazeri *et al.*, [66] also introduced an GAN based colourisation model building upon the work of pix2pix with some more improvements.

- Deoldify [5] is recent, image colourisation model based on GANs which uses state of the art techniques such as self-attention [103] and spectral normalisation [65] which seemed to improve the visual results substantially. In addition, the work also introduces a new GAN pre-training method called NoGAN which is trained based a perceptual loss.

4.4 Image Colourisation with Encoders and Generative Adversarial Networks

This section contrasts two different approaches to the image colourisation problem. This is done to discuss some of the challenges involved with image colourisation and to provide some context around the problem and delve into more technical details. In particular the following discusses

using an encoder based approach [104] by Zhang *et al.,* which uses a hand engineered loss function to achieve vibrant colourisation results, and an image translation method that uses Generative Adversarial Networks by [48] Isola *et al.*, which showed that reasonable image colourisation results can be achieved without hand engineering a complex loss functions. Additionally, this section also includes technical details to a self-attention module employed by state of the art GANs used for image synthesis [103].

4.4.1 Image Colourisation using an Encoder

The image colourisation Deep Learning model proposed by Zhang *et al.* [104] is a single Convolutional neural network Encoder model trained on the imageNet database [78] containing over 1.2 million images. The network architecture is based on the VGG convolutional neural network model (described in Section: 2.8.2 in Chapter 2) which was modified for colourisation. The model takes an input image's lightness channel (L) and predicts a corresponding a and b channels of the image in CIE Lab colour space. The final layer of the architecture is a softmax layer which outputs a distribution of quantized ab colour values at each pixel location of the image.

The Loss Function

Perhaps the most interesting part of this work is the loss function used to train the model which was tailor made to the colourisation problem. The loss function by Zhang *et al.* treats colourisation as a multinomial classification problem. When the ImageNet training data of over 1.2 million training images, are quantized by the *ab* values pairs by grid size 10, 313 spaces of quantized *ab* values pair bins are in gamut, thus making the colourisation problem a Classification problem of a 313-way prediction at each point (pixel) location of the output image. For a given image's lightness (L) channel X, a mapping to a probability distribution over possible colours $\hat{Z} = \mathcal{G}(X)$ are learnt where $\hat{Z} = [0, 1]^{H \times W \times Q}$ and Q is the 313 possible classes (of quantized ab value bins).

To compare the ground truth image Y against the prediction $\hat{Z}$, an additional function $Z = \mathcal{H}_{gt}^{-1}(Y)$, is used where a soft-encoding scheme converts the ground truth colours Y into the vector

Z, by finding a weighted 5-nearest neighbours to the ground truth colour $Y_{h,w}$ in the output space by taking the 5 closest colours and their proportional distance from the ground truth using a Gaussian kernel with $\sigma = 5$. The loss function is therefore a multinomial cross entropy loss function L_{cl} which was defined as:

$$L_{cl}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} log(\hat{Z}_{h,w,q}) \tag{4.1}$$

where $v(.)$ is a weighting term which is used to rebalance the loss based on colour's rarity based on the distribution of ab values. Empirically it was shown that in natural images, the ab values tend to be severely biased towards low values. This results in the imbalance of class which presents a problem when optimising a classification loss. Therefore, in order to account for this, the colour class are re-weighted during training time to emphasize rarer colours, where each pixel is weighted by a factor $\mathrm{w} \in \mathbb{R}^Q$ based on the closest ab bin as follows:

$$v(Z_{h,w}) = \mathrm{w}_{q^*}, q^* = \underset{q}{\mathrm{argmax}} Z_{h,w,q} \tag{4.2}$$

$$\mathrm{w} \propto \left((1-\lambda)\tilde{p} + \frac{\lambda}{Q}\right)^{-1}, \mathbf{E}[\mathrm{w}] = \sum_{q} \tilde{p}_q \mathrm{w}_q = 1 \tag{4.3}$$

where $\tilde{p} \in \bigtriangleup^Q$ is a smoothed empirical distribution, which is obtained by smoothing the distribution of the empirical probability of colours in the quantized ab space $p \in \bigtriangleup^Q$ with a Gaussian kernel G_σ. The the distribution is then mixed with a uniform distribution $\lambda \in [0, 1]$, where the reciprocal is taken and normalized so the weighting factor is 1 on expectation.

Predictions Made at Inference

At inference time, another additional function $\hat{Y} = \mathcal{H}(\hat{Z})$ is used to map the predicted distribution $\hat{Z}$ (i.e. the network's final softmax layer predictions) to a pixel location in ab space to produce the

final output image $\hat{Y}$. The function $\mathcal{H}$ is necessary to balance the trade-off between final output results which have spatial consistency and desaturated colour predictions when mapping from the $\hat{Z}$ distribution to a final output image. This is because it was found that taking the mode of the softmax layer's predicted distribution (i.e the colour with the highest activation) for each pixel results in spatially inconsistent results whereas taking the mean of the softmax layer's distribution results in output images that have desaturated colours due to the effect of averaging. Thus, the function $\mathcal{H}$ is used to alter the distribution of the softmax layer predictions $\hat{z}$, from which the mean is taken from that distribution to produce the final end results. This function was described as:

$$\mathcal{H}(\hat{Z}_{h,w}) = \mathbf{E}[f_T(\hat{Z}_{h,w})],\ f_T(\hat{z}) = \frac{\exp(\log(\hat{z})/T)}{\sum_q \exp(\log(\hat{z}_q)/T)} \tag{4.4}$$

where T is a temperature hyper-parameter. Setting $T = 1$ leaves the distribution of $\hat{Z}$ unchanged, lowering T results in a more sharply peaked distributions of activations where $T \to 0$ results in the output producing a one-hot vector representation. Thus, the final system is composed of the Convolutinal Neural network $\mathcal{G}$ which predicts a distribution of colours over all the pixels for the output image and $\mathcal{H}$ which produces the final output which is spatially smoothed.

Image Colourisation with Encoders: Limitations

The Encoder approach described thus far are able to achieve results which have vibrant colours and was an improvement over previous results at the time of its publication. Despite the merits of the above system, there are several downsides to the Encoder based approach, discussed below.

First, is the limitations of the system with respect to the visual quality. These are best described by presenting a few visual examples. Figures 4.1 and 4.2, each shows a comparison between the original images and examples of output results for the re-colourised images from the Author's replication [2] of the original work, trained from scratch on a subset of the ImageNet.

Figure 4.1. Example output (from a validation set) of colourisation using the encoder approach: An image of an object - the image on the left is generated, and the right image is the original

Figure 4.2. Example output (from a validation set) of colourisation using the encoder approach:: An image of an indoor structure - the image on the left is generated, and the right image is the original

Figure 4.1 shows an object colourised in red which lacks spacial consistency. The lack of spatial consistency despite the built-in spatial smoothing of the outputs done though the function $\mathcal{H}$. Figure 4.2, shows a particularly poor result for an indoor scene wall, with the confusion resulting red and blue artefacts being introduced in the image, (in addition to some other objects incorrectly coloured red) this is perhaps an indication of some difficulty in inferring structures which span the image globally, hence different parts of the walls are colourised with patches of red and blue colours. Some improvements for these common problems are further discussed in Section 4.5.

Second, is the limitation of hand engineering a suitable tailored loss function to colourisation. While hand engineering a loss function such as the kind described above can improve the visual quality of image colourisation, it would seem more desirable to allow a network to discover a

suitable loss function automatically. To that end, the Generative Adversarial Networks may provide a better path to improvement as loss functions are learnt through training.

4.4.2 Image Colourisation using Conditional Generative Adversarial Networks

Whereas the encoder approach learns a mapping where each output pixel is independent from each other, GANs are able to learn a structured loss. Isola *et al.,* [48] popularised the **conditional Generative Adversarial network (cGAN)** as a general propose image to image translation system. In Isola *et al.,*'s work [48], it is demonstrated that visually compelling image synthesis can be achieved without using a hand-engineering loss functions, by introducing a simple framework known as Pix2pix. The distinction of a cGAN compared to a standard GAN can be described as follows: whereas a GAN (see section 2.10.3) learns a mapping from a random noise vector to a target output y, $G : \{z\} \rightarrow y$, a cGAN in contrast, learns a mapping from an observed input x and a random noise vector z to y, $G : \{x, z\} \rightarrow y$.

The Conditional Generative Adversarial Network Objective

The Pix2pix framework [48] trains a cGAN which observes an input image x and uses a generator network G which produces output images that are difficult to distinguish from the 'real' images by using a discriminator network D. The objective for the framework proposed by Isola *et al.,* was described as:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))] \tag{4.5}$$

where G is optimised to minimise the objective against the adversarial network D which tries to maximize it. In addition, the generator is also optimised to be close to the ground truth 'real' images by mixing the $L1$ loss to the objective such that:

$$G^* = arg\ min_G max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G), \tag{4.6}$$

where the λ is a hyper parameter used to control the strength of the L2 objective, and

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y}[\|y - G(x)\|_1], \tag{4.7}$$

i.e., it is the $L1$ pixel wise distance between the generator network's output and the 'real' image. Overall, the objective described above allows differences between the generated images and the 'real' images to be penalised such that the low-frequency structures of images are penalised by the $L1$ objective while the discriminator's D network's objective enforces the correctness higher-frequency structures of the images.

The Architecture and Optimisation

The Discriminator network uses an architecture which consists of contracting layers which down-samples the input image to capture context, which is proceeded by and symmetric expanding layers which up-samples the images. In addition, skip connections [73] are added between each ith and $n - i$th layer where n is the total number of layers, and each skip connections simply concatenates all the channels at the two layers.

The Pix2pix frame work also showed that better quality of outputs are generated by restraining the evaluation of the Discriminator Network to local patches of images rather than evaluating on the whole image. This requires a special architecture for the discriminator network termed the **PatchGAN**. The PatchGAN discriminator network evaluates and classifies each $N \times N$ patch from the generator network's output, convolutionally where they are averaged across the whole image. This allows the discriminator to evaluate on local structures of the generated image instead of the image as a whole.

The optimisation follows the standard GAN approach where training is alternated between one

gradient descent step on the Discriminator, then one step on Generator. Training is done using SGD with the Adam solver [49] for both networks.

4.4.3 Self-attention Mechanism for Image Synthesis using GANs [103]

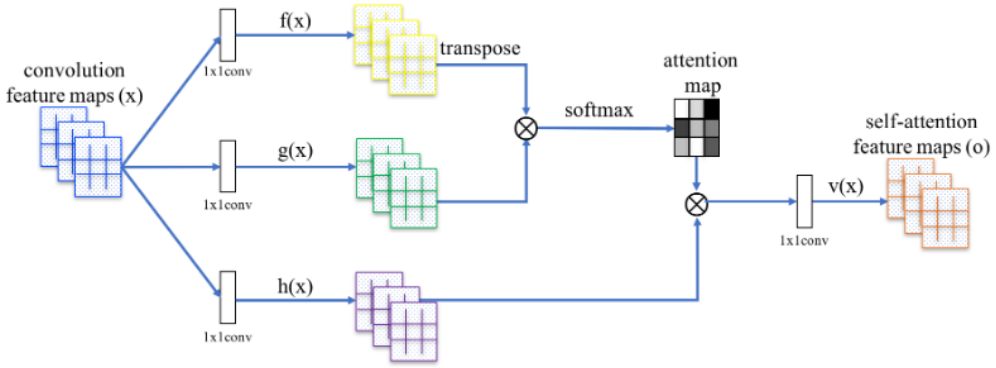


Figure 4.3. Self-attention module for the SAGAN (Image source [103])

The following provides details for the self-attention module introduced by Zhang *et al.,* [103]. Briefly, the self-attention module calculates a response at a point as a weighted sum of the features of all positions. This self-attention module is shown in Figure 4.3, where $\otimes$ denotes a matrix multiplication. The self-attention module are layers in a generative adversarial networks. The SAGAN attention modules takes the image input features from the previous hidden layer x which are transformed into two feature spaces f, g where $f(x) = \mathbf{W}_f x$, $g(x) = \mathbf{W}_g x$. These two feature spaces are then used to calculate the attention by:

$$\beta_{j,i} = \frac{\exp(s_{i,j})}{\sum_{i=1}^{N} \exp(s_{i,j})}, \text{where } s_{i,j} = f(x_i)^T g(x_j), \tag{4.8}$$

$\beta_{j,i}$ therefore indicates the extent the model attends to the ith location when synthesising the jth region. The output of the attention layer is then given as $o = (o_1, o_2, ..., o_j, ..., o_N) \in \mathbb{R}^{C \times N}$, where:

$$o_j = v(\sum_{i=1}^{N} \beta_{j,i}), h(x_i) = \mathbf{W}_h x_i, v(x_i) = \mathbf{W}_v x_i \tag{4.9}$$

where, $\mathbf{W}_g \in \mathbb{R}^{\bar{C}\times C}$, $\mathbf{W}_f \in \mathbb{R}^{\bar{C}\times C}$, $\mathbf{W}_h \in \mathbb{R}^{\bar{C}\times C}$ and $\mathbf{W}_v \in \mathbb{R}^{C\times\bar{C}}$ which are learned weight matrices by 1×1 convolutions, and the channel numbers $\bar{C}$ is reduced by $\frac{C}{k}$ where $k = 1, 2, 4, 8$ (i.e., if $k = 8$, then $\bar{C} = \frac{C}{8}$) for memory efficiency. Lastly, the final output is obtained by multiplying o with a learnable scaling parameter γ and adding the input as a residual connection, therefore the final output is given by:

$$y_i = \gamma o_i + x_i. \tag{4.10}$$

4.5 The Proposed Method

This work borrows from the Pix2pix framework and treats image colourisation as a image translation problem while building on crucial insights from state of the art in image synthesis using GANs to improve output quality. Additionally, the network architecture implemented in this work is computationally resource light, and is streamlined to reduce memory requirements and training time. The method can be summarised as having the following advantages:

- **Learning a structured loss using generative adversarial networks -** The framework offered by Pix2pix allows for a structured loss to the colourisation problem compared to a pixel-wise, unstructured losses used by encoder approaches where each output pixel is learned conditionally and independently from the others in a given input image. Furthermore, given that with Deep Learning, features are no longer engineered, but are automatically discovered through training, it is likewise perhaps more practical to adapt a GAN based approach which learns the mapping and loss function through training. This eliminates the need of hand-engineering a loss function for colourisation. This shifts the problem from engineering a tailored loss to discovering more suitable models and implementing better training to achieve better colouration results.

- **Colourisation with better spatial modelling using attention -** Conventional convolution layers seem to have difficulty in modelling long-range dependencies, and synthesising images

with complex geometries which have many structural constraints, compared to images which have few structural constraints such as images of landscapes. Zhang *et al.,* [103] pointed out the limitations of using conventional convolution layers for image synthesis because of the difficulties of modelling dependencies across spatially far apart regions of the images, due to the small sizes of the convolutional operator's local receptive fields. As a result, with conventional convolution layers, details in images are generated as a function of spatially local points from a low resolution feature maps. To better model long-range dependencies the Self-Attention Generative Adversarial Networks [103] introduced the self-attention module which work complementary to the operations in convolutional layers. Since the successful image colouration hinges on the model's spatial coherence and modelling of long-range dependencies self-attention modules provide a path for improvement.

- **Efficient architecture design and training -** To optimise for efficiency, the proposed model uses no more than 4.14M parameters in total for both the discriminator and generator network, which is achieved by employing depth-wise separable convolutions (see Section: 2.8.1) which allows for computation efficiency, and the reduction of memory requirement. In addition the model is trained by applying spectral normalisation (introduced by Miyato *et al.,* [65]) to both the discriminator network and the generator network, as suggested by Zhang *et al.,* which sets the spectral norm of all the weight layers to 1. Spectral normalisation require no extra hyper-parameter tuning, and require little additional computational cost.

4.5.1 Architecture Details

The system has a network architecture as shown in Figure 4.4. The **generator** architecture has an encoder-decoder structure and formed of 14 Res-block [35] like modules which uses depth-wise separable convolutions (see Section: 2.8.1). The number of channels allocated for each res-block in the encoder-decoder structure is as follows: encoder uses [64, 128, 256, 256, 256, 256, 256]channels, while the decoder uses [256, 256, 256, 256, 256, 128, 64]. The generator network also uses U-net [73] skip-connections which concatenates the features between the ith and $n - i$th for the $i > 3$ Res-blocks, where n is the number of the total res-blocks. In between the encoder and the decoder

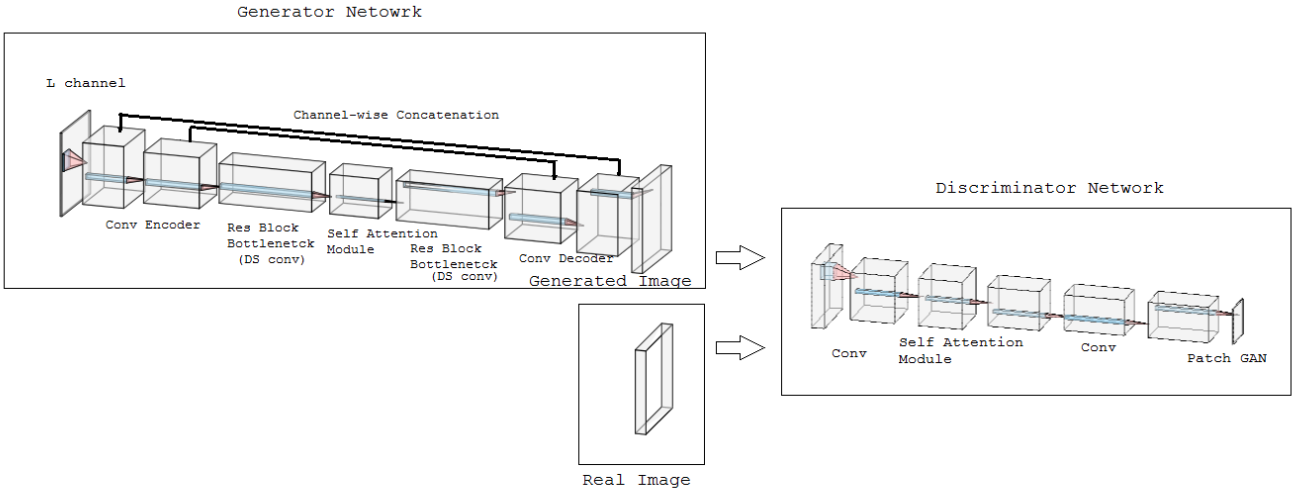


Figure 4.4. The proposed Efficient Image Colourising GAN (this figure is not exact to the scale of parameter sizes)

of the generator, the self-attention module (as described in Section 4.4.3) was inserted with 256 channels to handle the modelling of long range dependency. Additionally, the architecture has initial layer which uses convolution-BatchNorm-ReLU layer and a final convolution layer which uses convolution-BatchNorm-Tanh which produces the final output colourised image.

The **discriminator** network has a PatchGAN architecture (see Section: 4.4.2) where the discriminator tries to classify if a patch region of an image is generated by the generator network (i.e., fake), or sampled from the ground truth (real) image with colours . The discriminator consists of an initial convolution layer with 64 channels immediately followed by a self-attention module with 64 channels, which is then followed by a series of 3 res-blocks which each have double the number of channels from the previous res-block, with a final PatchGAN classier of size 12×12 pixels. All convolution layers used in the discriminator uses convolution-BatchNorm-leakyRelu with slope value of 0.2

4.5.2 Training Details and Optimisation

For training data, the unlabelled set from the STL10 dataset was used which contain 100,000 images which has sizes of 96×96 pixels. The discriminator and generator networks are trained on the Pix2pix loss objective as described in Section: 4.4.2, and follows the standard GAN optimisation procedure by alternating between one stochastic gradient descent step of the Discriminator and one step of the Generator. The Adam optimiser was used with a learning rate of 2e-4 for 5 epochs, with batch sizes of 128.

4.6 Experiments and Results

This section provides some experimental results to provide some quantitative assessment of the method proposed against several alternative models. The experiment consists of evaluating the predictions of the models for 8000 unseen images from the STL10 test set. Henceforth, referring to the proposed model as the **cGAN+Att+DSConv+SN** model, this model is compared against the following alternative models:

- **cGAN.** A conditional generative adversarial network for image colourisation. Unlike the proposed model, the network architecture in this implementation does not include the self-attention modules or depth wise separable convolution layers.
- **cGAN+Att.** A conditional generative adversarial network with standard conovlution layers which uses self-attention modules in the same layers as the proposed model.
- **cGAN+Att+DSConv.** A conditional generative adversarial network with self-attention modules and depth wise separable convolution i.e., it a model with a network architecture that is identical to the proposed model. However, unlike the proposed model, it is trained without applying the spectral normalisation [103].

To keep the comparison as fair as possible, all models are trained with the same training parameters as the proposed model. The models are evaluated for the following key metrics:

- **Average pixel RMSE (RGB)-** The pixel wise average euclidean distance between the colourisation predictions of the model and the original colour images in RGB space.
- **Average pixel dE Lab -** The pixel wise average colour difference between colourisation predictions of the model and the original colour images in Lab space.
- **Total parameters -** The total number of parameters required for the GAN architectures.
- **Train time -** Total training time taken to complete 5 epochs of training.

Method	RMSE (RGB)	dE Lab	Total parameters	Train time
cGAN	30.69	11.64	11,740,484	21291
cGAN+Att	26.87	10.59	11,874,089	24360
cGAN+Att+DS Conv	29.32	11.04	**4,135,798**	**17018**
cGAN+Att+DS Conv+SN(Proposed)	**25.51**	**10.38**	**4,135,798**	17144

Table 4.1. Comparison of Image Colourising GAN models using qualitative measures

Table 4.1 shows the results for each model for the above metrics. It should be noted that the results for the RMSE and dE Lab, which serves as a proxy for assessing the Image colourisation quality are generally quite high in values across all the models. This is somewhat expected since this is not strictly a image reconstruction task, but rather, the networks are predicting plausible coloured version of the image. Nonetheless, against the other models tested, the proposed model produces the lowest average pixel wise RMSE and dE Lab while using the least amount of total parameters and reasonable training time. With the input image resolution of 96×96 pixel, whereas a standard cGAN architecture requires approximately 11.7M total parameters using standard convolution layers, models which use depth wise separable convolutions are able to reduce the required total parameters to approximately 4.1M, in addition to significantly reducing training time for computing the same amount of iterations. The implementation of spectral normalisation during training produces the best results while requiring little additional computation.

Additionally, some samples of the outputs from the proposed system can be found in Appendix C to provide some visual samples of the outputs. One open question that remains is that a small

number the of samples from the colourised images, appeared to suffer from a failure mode, where the outputs are left in grey-scale which have also been reportedly found to be common in the study by Isola *et al.,* [48].

4.7 Conclusion

This chapter explored the application of Deep Learning for image colourisation which has the potential to automate the traditionally labour intensive graphics task. Many popular early work of Deep Learning image colourisation involved the Encoder based approach, which predicts each output pixels independent from each other. Despite carefully defined loss functions, and efforts to manually spatially smooth the final outputs, it was found that the Encoder based approaches often produced colouration results that are spatially inconsistent. Thus, many later works in the literature seem to favour the use of generative adversarial networks which trains the networks to learn both the mapping function of the generator network and a structured loss through the discriminator network.

This work demonstrated that the complex task of image colourisation can be achieved with a network architecture streamlined for efficient using factorised convolution layers to significantly reduce parameter size and computation cost while achieving visually good results. The implementation in this work adapted the GAN approach to image colourisation, and builds on insights from state of the art GAN models, and in particular recently introduced training techniques and efficient factorised convolution to reduce excessive computation requirements and while improving colourisation results. The use of self-attention modules are explored to better model long-range spatial dependency and thus provided some improvements over using of using conventional convolution layers. In addition, more recent training techniques were used such as the Spectral Normalisation, which when applied during training has also found to further improve the colourisation quality.

Chapter 5
Supervised Deep Learning and Colours: Colour Spaces and Attention for Image Classification

5.1 Introduction and Motivation

The objective of this chapter is to explore colour spaces as a means for improving Convolutional Neural Networks for the supervised learning setting. The motivation for this study comes from the fact that most modern deep Convolutional Neural Networks are simply trained on the RGB colour components of the input image. By using the various transformation formulae (described in Chapter 2.12), it is possible to convert between the colour spaces, to transform the input representation. The purpose of this study is to investigate if the various colour space transformations would result in performance difference for the task of image classification with Convolutional Neural Networks.

5.2 Related Work

There are ample amounts of examples which colour space transformations as a preprocessing step as they have been commonly used as parts traditional machine learning pipelines [11, 89, 90]. In more recent works which uses Deep Learning, colour space conversion are treated as engineered solutions for very specific application needs. For example a recent study [10] applied HSV colour space transformation to the inputs be used as a spatial image segmentation step which are fed to

neural network that uses the LeNet5 [59] architecture for the tasks of traffic sign detection and recognition. If not for a specific application needs, generally Convolutional Neural Networks, use the RGB colour spaces as inputs.

However, there are few studies which used more than just the RGB colour space for image classification tasks and achieved performance boots. An example is a recent study by Gowda *et al.,* [32] which proposed a system which uses an ensemble of DenseNets models which each uses a different colour spaces as inputs.

5.3 Experiment 1: Alternative Colour Spaces for Convolutional Neural Networks

As a first experiment, the following aims to find out if converting between different colour spaces, to transform the input representation using the various colour space transformation equations will result in any notable difference in performance for the Convolutional neural networks. This is done by converting the original RGB images into different colour spaces of LAB, HSV, YUV, and XYZ.

Another alternative would be to combine different colour spaces by stacking the colour components of multiple colour spaces channel-wise as input (e.g., RGB and LAB) and increasing the number of input node channels for the Convolutional neural network as needed by the additional channels for the colour spaces.

To test the methods above, two modern architectures of Convolutal neural network models (Densenet and Mobilenet) were selected to be trained on the popular Cifar 10 and 100 benchmarking datasets [52]. The two models were chosen arbitrarily, but they provide a comparison for different kinds for networks. The full training details could be found in the Appendix (see Appendix B.0.2). Each models were individually trained on the LAB, HSV, YUV, and XYZ colour spaces and creating different combinations of 'stacked' multi colour spaces by combining different combinations colour components from the RGB, LAB, HSV, YUV colour spaces. These models

were evaluated by the standard protocol of the Cifar dataset i.e., the classification performance of the trained models on the predictions for test set. [52].

5.3.1 Results

	Densenet121		Mobilenet224	
	Cifar 10+	Cifar 100+	Cifar 10+	Cifar 100+
RGB(baseline)	94.84	76.71	93.66	73.61
LAB	94.18 (-0.66)	74.4 (-2.31)	92.69 (-0.97)	71.3 (-2.31)
HSV	94.95 (0.11)	75.85 (-0.86)	**93.78 (0.12)**	**73.31 (-0.3)**
YUV	**95.1 (0.26)**	**76.26 (-0.45)**	93.46 (-0.2)	73.22 (-0.39)
XYZ	94.68 (-0.16)	75.65 (-1.06)	93.32 (-0.34)	72.75 (-0.86)
Multi Colour spaces				
RGB, LAB	95.09 (0.25)	**77.04 (0.33)**	**93.92 (0.26)**	73.32 (-0.29)
RGB, LAB, HSV	94.76 (-0.08)	76.53 (-0.18)	93.8 (0.14)	73.31 (-0.3)
RGB, LAB, HSV, YUV	**95.11 (0.27)**	76.98 (0.27)	93.86 (0.2)	**73.84 (0.23)**

Table 5.1. Performance (accuracy) comparison for Densenet and Mobilenet on different colour spaces against baseline RGB the best performance for each category is highlighted in bold font

The Table 5.1 shows the results of the classification accuracy performance for text set in percentage. The results does not seem to show meaningful patterns and the use of different colour spaces of Convolauional neural networks only results in minute difference in performance.

5.4 Hybrid Colour Spaces using Attention

A hybrid colour spaces is the idea of either interactively or adaptively combining different colour components from different colour spaces to increase the effectiveness of the colour components to discriminate colour data, and to reduce the rate of correlation between colour components [89, 19]. The use of hybrid colour spaces specifically for Convolutional Neural Network seem to be sparse in the literature. The challenge of using a hybrid colour space is determining relevant colour components. Previous methods for for finding optimal colour component combination for the hybrid colour space include the use of generic algorithm heuristics and principal component analysis [68] which adds complexity to the process.

The following attempts to explore an automatic method of determining a the relevant colour components for a hybrid colour space specific to a convolutional network architecture by using the self-attention modules, since in theory the self-attention modules in convolutional neural networks learn a **vector of importance weights** by using gating mechanisms. The Squeeze-and-Excitation module (see section 2.11.2) is an example of a popular self-attention mechanism which uses a gating mechanism .

By using the attention modules from the Squeeze-and-Excitation networks (see section 2.11.2) in the first layer of network architecture and observing vector of activations of the attention modules some patterns was found to emerge, it is hypothesised that these reveal some amount of colour component importance. These colour component importance can then be used to determine the relevant colour components to create a hybrid colour space of a convoltuional neural network architecture.

Attention Hybrid Colour Spaces

The method for building hybrid colour spaces using the self-attention mechanism is implemented as follows:

1. Convolutional network model of a model architecture which uses the Squeeze and Excitation module is trained on a data set. The Convolutional network model has an arbitrary number of colour channels, call this number $\mathbf{C}$.

2. Once the model is trained, the forward pass is computed on a large number of images, and the activations of the vector $\mathbf{s}$ (in Equation 2.44) form the Squeeze and Excitation module in the first layer is recorded.

3. The recorded activations are averaged. The averaged activations is a vector, which is the same length as the number of input channels of Convolutional network model. Each element in the vector corresponds to an input channel of Convolutional neural network input i.e, a single colour component.

4. using the averaged pattern of activations by ranking the corresponding colour components with the strongest activations first, a hybrid hybrid colour spaces which uses the number of colour components $< \mathbf{C}$ is created for the model architecture by discarding the colour components corresponding to the lowest ranking activations.

Using the above process for the DenseNet and the MobileNet for the cifar10 and cifar100 dataset with $\mathbf{C} = 12$, produces the results in shown Table 5.2.

Colour Component Ranking by Highest Activation				
	DenseNet		MobileNet	
Channel	Cifar10	Cifar100	Cifar10	Cifar100
RGB(R)	10	7	11	10
RGB(G)	7	2	9	8
RGB(B)	2	1	7	11
LAB(L)	1	6	8	7
LAB(A)	5	8	5	3
LAB(B)	8	4	2	4
HSV(H)	12	10	3	1
HSV(S)	11	11	6	2
HSV(V)	3	12	12	12
YUV(Y)	6	5	10	6
YUV(U)	9	9	4	5
YUV(V)	4	3	1	9

Table 5.2. A colour component ranking based attention activation values for each task and architecture

Using Table 5.2 to create a hybrid colour space for DenseNet with 3 colour components for Cifar10 would result in colour components in the order of L (from LAB), B (from RGB) and V (from HSV) as these were the ranking order of the highest activation values. The rankings from table 5.2 was then used to create Convolutional network with hybrid colour spaces with 6 and 9 colour component channels for the classification task of Cifar10 and Cifar100. The results for these models on the classification accuracy performance for test set in percentage and compared against RGB baselines from the previous experiment is shown in Tab 5.3. Lastly, the averaged activation patterns for the different hybrid colour spaces, for different datasets and models can be found on Appendix D these are included only for reference as it does not provide any more information than the individual activation patterns for different models for different Datasets.

	Densenet121		Mobilenet224	
	Cifar 10+	Cifar 100+	Cifar 10+	Cifar 100+
Attention & Hybrid Colour Spaces				
12-channels (With Colour Attention)	95.10 (0.26)	76.90 (0.19)	94.04 (0.38)	73.60 (-0.01)
Top 6-channel hybrid	94.97 (0.13)	77.07 (0.36)	**94.15 (0.49)**	**73.78 (0.17**)
Top 9-channel hybrid	**95.11 (0.27)**	**77.17 (0.46)**	94.05(0.39)	73.65 (0.04)

Table 5.3. Comparison of classification accuracy for Densenet and Mobilenet using hybrid colour spaces

5.5 Conclusion

This chapter explored the utility of different colour spaces other than RGB as a means for improving Convolutional Neural Networks. First an experiment comparing colour spaces for different architectures, was conducted, which also included multi colour space which uses a combination of different colour spaces which are fed as inputs to the Convolutional Neural Networks. In the experiment conducted it was found that the uses of different colour spaces only results in minute differences in performance if any gains are made (no more than half-percent gains were mode), though most colour spaces perform worst off than RGB. It is therefore concluded that at least for classification tasks for convolutional neural networks, using the RGB colour space is a safe choice.

This chapter then introduced a method which purportedly creates hybrid colour spaces automatically using Self-attention mechanism which are specific to different architectures and tasks (datasets). Experimentally, the initial results seem to be no worst than using the different non-RGB colour spaces, although some more thorough analysis would be required for verifying this method.

Chapter 6

Conclusion

6.1 Thesis Summary

This thesis was written in the hopes of making a small but meaningful contribution towards the research areas which concern Deep Learning for vision and colours.

Chapter 2 provided an overall literature review for Artificial Neural Networks and Deep Learning. This chapter began with an introduction to machine learning, and the various learning paradigms. Then, a historical context an Artificial Neural Network as provided, where it was stated that one of the long standing challenges that remained for Deep Learning is making the Deep Learning algorithms learn without using human annotated data. Following that, details for the Multi-layer perceptron, Convolutional Neural Networks and details for more advanced concepts were provided. Thereafter, several methods for unsupervised learning with Deep Learning was discussed and some details on Attention Mechanisms for Deep Learning was provided. Lastly, a brief summary on colour concepts were provided and in particular described some of the mathematics for transforming colour spaces.

In chapter 3 the aim was to explore methods of self supervised learning for visual representations and how colours could play a role in achieving this. This chapter began by providing a brief review of works in the literature which relate to self supervised learning while building some context around them. Following that, this chapter investigated the self supervised learning methods which involve the predictive task of image colourisation, leading to an implementation of one of the methods, which lead to the conclusion that in practice, using colour predictions alone as

a predictive task for self supervised learning is not sufficient, likely because the representations learnt by them do not generalise well enough to other tasks. This motivated the investigation of other methods for visual representation learning and in particular a method known as contrastive learning. This lead to the development of a self supervised learning method which combines both the colour prediction task with the contrastive learning method which delivered some promising results as a method improving visual representations.

Chapter 4 explored the application of Deep Learning for the task of image colourisation from a graphics task perspective, which has the potential to automate this tremendously labour intensive task usually performed by human artists. This chapter provided a brief survey of literature for the related work and found that the earlier approaches to this problem which used Convolutional Neural Network models often used carefully defined loss functions and manually smoothed the final outputs, yet despite this, was often fell short of producing spatially consistent results and satisfactory outputs. It was hypothesised that these were in part due to the limitations of the conventional convolution layers and the encoder network based approach. Therefore, the implementation in this work adapted the Generative Adversarial Network approach while building on insights from recently introduced Deep Learning models with the aim of improving colourisation quality. The final model proposed used a streamlined architecture, with efficient convolution layers which demonstrated that the a task as complex as image colourisation can be achieved with significantly low computation cost while producing visually good results, this was found by comparing qualitative results with several other baseline models.

Chapter 5 investigated colour space Deep Convolutional Neural Networks classification models, and originally sought to find if performance gains are available using different colour spaces for Different models by searching different combinations to however, it was concluded that in most cases the differences in performance were minute. Instead, some experiments were conducted by using attention mechanism to observe patterns of internal activations in the Convolutional Neural Networks.

6.2 Future Work and Limitations of this Study

Finally this section summaries some directions of future work for this study:

- In chapter 3 it was noted that the experiments in was limited by the fact that the visual representations are evaluated only for the downstream classification task, and therefore expanding variety of downstream task for evaluation is likely to benefit this study. Additionally, given that it is claimed that contrastive learning benefits from larger batch sizes and longer training [13], a larger scale implementation to find the upper-bounds limits of performance for the method proposed is a future work.
- In chapter 4, in the context of image colourisation, there is currently is no standard evaluation metric for quantifying the quality of colourisation models despite there being evaluation metrics available of for tasks such as image synthesis using generative models [8]. Therefore, formulating one would benefit this work.
- In chapter 5, a novel concept was introduced, however these were based on intuition and empirical results only, some more through analysis would be required to prove its validity.

Lastly, as a general note, many of the experiments in this study was done on moderate computational resources which presents clear limitations as a Deep Learning research. Nonetheless the experiments presented in the sections above were designed with the intention to provide valid results for the limited implementation scale.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Akanuma. Transfer report - fine inference deep learning models for image colourization. 2017.

[3] I. Aleksander and H. Morton. *Aristotle's Laptop: The Discovery of Our Informational Mind*, volume 1. World Scientific, 2012.

[4] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. C. V. Esesn, A. A. S. Awwal, and V. K. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *CoRR*, abs/1803.01164, 2018.

[5] J. Antic. Deoldify https://github.com/jantic/deoldify, 2019.

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for scene segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[7] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[8] S. Barratt and R. Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[9] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[10] J. Cao, C. Song, S. Peng, F. Xiao, and S. Song. Improved traffic sign detection and recognition algorithm for intelligent vehicles. *Sensors*, 19(18):4021, 2019.

[11] D. Chai and A. Bouzerdoum. A bayesian approach to skin color classification in ycbcr color space. In *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No. 00CH37119)*, volume 2, pages 421–424. IEEE, 2000.

[12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[13] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[14] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng. Dual path networks. *CoRR*, abs/1707.01629, 2017.

[15] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[16] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[17] A. K. R. Choudhury. *Principles of colour and appearance measurement: Object appearance, colour perception and instrumental measurement*. Elsevier, 2014.

[18] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[19] P. Colantoni and Al. Color space transformations. 2004.

[20] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995.

[21] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.

[22] R. Dahl. Automatic Colorization http://tinyclouds.org/colorize/, 2016.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[24] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[25] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[26] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning, 2016.

[27] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1:119–130, 1988.

[28] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[29] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[30] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[32] S. N. Gowda and C. Yuan. Colornet: Investigating the importance of color spaces for image classification. In *Asian Conference on Computer Vision*, pages 581–596. Springer, 2018.

[33] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[34] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[37] D. O. Hebb. *The organization of behaviour*. 1961.

[38] G. E. Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[39] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[40] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[41] T. K. Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society.

[42] J. J. Hopfield and D. W. Tank. neural computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[44] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[45] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[46] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (TOG)*, 35(4):110, 2016.

[47] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[48] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2016.

[49] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[50] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[51] D. P. Kingma and M. Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, 2014.

[52] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.

[55] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision*, pages 577–593. Springer, 2016.

[56] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011.

[57] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–, May 2015.

[58] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[59] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[60] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[61] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[62] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[63] M. Minsky and S. Papert. Perceptrons. 1969.

[64] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[65] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[66] K. Nazeri, E. Ng, and M. Ebrahimi. Image colorization using generative adversarial networks. *Lecture Notes in Computer Science*, page 8594, 2018.

[67] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *CoRR*, abs/1603.09246, 2016.

[68] M. M. Oghaz, M. A. Maarof, A. Zainal, M. F. Rohani, and S. H. Yaghoubyan. A hybrid color space for skin detection using genetic algorithm heuristic search and principal component analysis technique. *PloS one*, 10(8):e0134828, 2015.

[69] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[70] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1):145–151, Jan. 1999.

[71] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2016.

[72] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2018.

[73] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[74] F. Rosenblatt. Principles of neurodynamics. 1962.

[75] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[77] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[78] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[79] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[80] L. Sifre and S. Mallat. Rigid-motion scattering for texture classification. *CoRR*, abs/1403.1687, 2014.

[81] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[82] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[83] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[84] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[85] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[86] D. Swanston, J. Bishop, and R. Mitchell. Simple adaptive momentum. 30(18):1498–1500, 1994.

[87] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[88] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. 2019.

[89] N. Vandenbroucke, L. Macaire, and J.-G. Postaire. Color pixels classification in an hybrid color space. volume 1, pages 176 – 180 vol.1, 11 1998.

[90] N. Vandenbroucke, L. Macaire, and J.-G. Postaire. Color image segmentation by pixel classification in an adapted hybrid color space. application to soccer image analysis. *Computer Vision and Image Understanding*, 90(2):190–216, 2003.

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[92] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.

[93] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[94] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

[95] M. R. William T. Freeman. Orientation histograms for hand gesture recognition. Technical Report TR94-03, MERL - Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, Dec. 1994.

[96] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. 2016.

[97] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[98] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks?, 2014.

[99] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[100] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[101] M. D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[102] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *In CVPR*, 2010.

[103] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363. PMLR, 09–15 Jun 2019.

[104] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[105] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. *CoRR*, abs/1611.09842, 2016.

[106] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction, 2016.

[107] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

Appendix A

Stochastic Gradient Descent Optimisers

This section includes a more detailed explanation for the various grandest decent variants. Many of the explanations comes from the excellent original source [75] which the reader is referred to for more details.

A.0.1 Gradient Descent

Gradient descent minimises an objective loss function $J(\theta)$ parametrised by the model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction to the gradient of the loss function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. This involves iteratively taking steps of a determined size set by a learning rate η as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Importantly, this involves following the direction of the slope of the surface created by the loss function to a local minima. However, as raised by Ruder [75] the basic gradient descent in its form noted above are faced with the following challenges:

1. Finding an appropriate learning rate is difficult as optimising with a small learning rate leads to painfully slow convergence while a learning rate that is too large can hinder convergence to a solution since it can cause the loss function to fluctuate.

2. A dataset's characteristics are often unknown in advance, which makes it difficult to specify a learning schedule to appropriately adjust learning rate while learning in order to achieve

optimal learning.

3. Standard gradient decent applies a uniform learning rate to all parameters which can be problematic if data is sparse and have features with very different frequencies, making it difficult to learn from useful features that occur only rarely in the dataset.

Modern neural networks used for Deep Learning are optimised by some variants of the basic gradient descent algorithm with improvements motivated by the challenges noted above. Below are some of the gradient descent optimisation algorithm which have become popular in recent years.

A.0.2 Adagrad

Adagrad [25] is an algorithm that adapt the updates to each individual parameter. Therefore, they have adaptive learning rates for each parameter, allowing for larger updates for infrequent and smaller updates for frequently adjusted parameters improving the robustness of learning. Additionally, this makes Adagrad suitable for learning with sparse data such as word embeddings, as infrequent (rare) words require larger updates than frequent (commonly occurring) words.

Whereas momentum performed updates for all papramter θ for every parameter θ_i with the same learning rate η. Adagrad uses a per-parameter update which are vectorized. As shown in the following, setting $g_{t,i}$ to be the gradient of the loss function, w.r.t the the parameter θ_i at time step t:

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

The update for every parameter θ_i at each time step t becomes:

$$\theta_{t+1,i} = \theta_{t+1,i} - \eta \cdot g_{t,i}$$

Using this update rule, Adagrad modifies the overall learning rate η at each time step t for every

parameter $\theta_i$based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t+1,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

where $G \in \mathbb{R}^{d \times d}$, a diagonal matrix wgere each diagonal element i, i is the sum of squares of the gradients w.r.t. θ_i up to the step t. Additionally, ϵ is a smoothing term to avoid division by 0 (usually set to a small value such as 1e-8).

Thus, with G_t containing the sum of squares of the past gradients w.r.t to all parameters θ diagonal, the implementation is vectorised by performing an element-wise matrix-vector multiplication $\odot$ between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

A.0.3 Adadelta

Adagrad's weakness is the accumulation of the squared gradients in the denominator: the accumulated sum keeps growing during training which causes the learning rate to shrink and eventually become infinitesimally small. Adadelta [101] improves Adagrad by restricting the window of accumulated past gradients to a fixed size. In adadelta, the sum of gradients is recursively defined as a decaying average of all past squared gradients by maintaining the running average $E[g^2]_t$ at time step t which depends only on the previous average and the current gradient as follows:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

where γ is set to a similar value to the momentum term of 0.9. By rewriting the gradient descent update in terms of the parameter update vector as $\Delta\theta_t$ it gives the following:

$$\Delta\theta_t = -\eta \cdot g_{t,i}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Then, the previously derived parameter update vector of Adagrad takes the form:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

The diagonal matrix G_t is then replaced with the decaying average over past squared gradients $E[g^2]_t$:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Notice the denominator is just the root mean squared error criterion of the gradient, therefore it is replaced with the shorthand (RMS):

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

Note that since the units in the update do not match (GD, momentum or adagrad) the updates should have the same hypothetical units as the parameter, another exponentially decaying average is needs to defined for the squared parameter updates:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Replacing the learning rate η in the previous update rule with $[\Delta\theta]_t$ finally yields the Adadelta update rule:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[\Delta\theta]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

A.0.4 RMSprop

RMSprop is an unpublished adaptive learning rate method introduced by Geoffrey Hinton. It is identical to Adadelta, except Adadelta uses the additional RMS parameters for the updates in the numerator for its update rule where as RMSprop does not. Thus, similar to Adadelta:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Hinton suggests γ to be set to 0.9, learning rate η to be 0.001 and 10^{-8} for ϵ.

A.0.5 ADAM

Finally, the Adaptive Moment Estimation (Adam) [49] is yet another method that computes adaptive learning rates for each parameters. The Adam, like Adadelta and RMSprop, stores the

exponentially decaying average of past gradients v_t, and keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_1$$

$$v_t = \beta_2 m_{t-1} + (1 - \beta_2) g_1^2$$

Additionally, as m_t, v_t, are initialized as vectors of 0's, they are biased towards 0. These biases are counteracted by computing the bias corrected estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

These are used to update the parameters as seen with Adadelta and RMSprop, yielding the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

The authors gives default values of 0.9 for β_1^t, 0.999 for β_2^t and 10^{-8} for ϵ.

Appendix B

Additional Materials and Extended Results

B.0.1 Archetecture Details of Popular Models Featured in this Study

The following includes the details for the exact architectures of the models described in various parts of this thesis. To avoid errors, the tables below are taken from the exact original source for ResidualNets, DenseNets and MobileNets from [35], [45] and [43] respectively.

ResidualNets

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

DenseNets

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

B.0.2 Implementation Details of Models Trained in Chapter 6

This section provides implementation details for the models trained in the Experimental section. All models were trained from scratch and the architectures used were based on the Densenet121 and Mobilenet shown in Appendix B.0.1. To be more precise the Densenet used was a DenseNet-BC variant with parameters $k = 12$, which uses transition layers and compression ($\theta = 0.5$) [45] while the Mobilenet was implemented as standard. In addition, since these two architectures were originally designed for the ImageNet dataset which uses relatively high resolution images, some minor spacial adjustments were made to the networks to accommodate the Cifar dataset image sizes of 32×32 pixels. This was done by changing the first Conv layer from 7×7 sized filters to 3×3 sized filters and removing the first Max-pooling layer of the Densenet. The final classifier layers of the architectures where adjusted for the 10 and 100 units for the class labels for Cifar 10 and 100 respectively and no other changes were made architecturally unless stated otherwise specifically, in the Chapter.

Both models were trained with SGD using batch sizes of 32 for 300 epochs and using a training schedule that divides the initial learning rate by 10 upon completing 50% and 75% of the total number of training epochs. The Densenet uses an initial learning rate of 0.1, a weight decay of 2e-4 and nesterov momentum of 0.9 while the mobilenet uses an initial learning rate of 0.01, a weight decay of 5e-4 and nesterov momentum of 0.9. Both models uses the Xavier weight initialisation method [29].

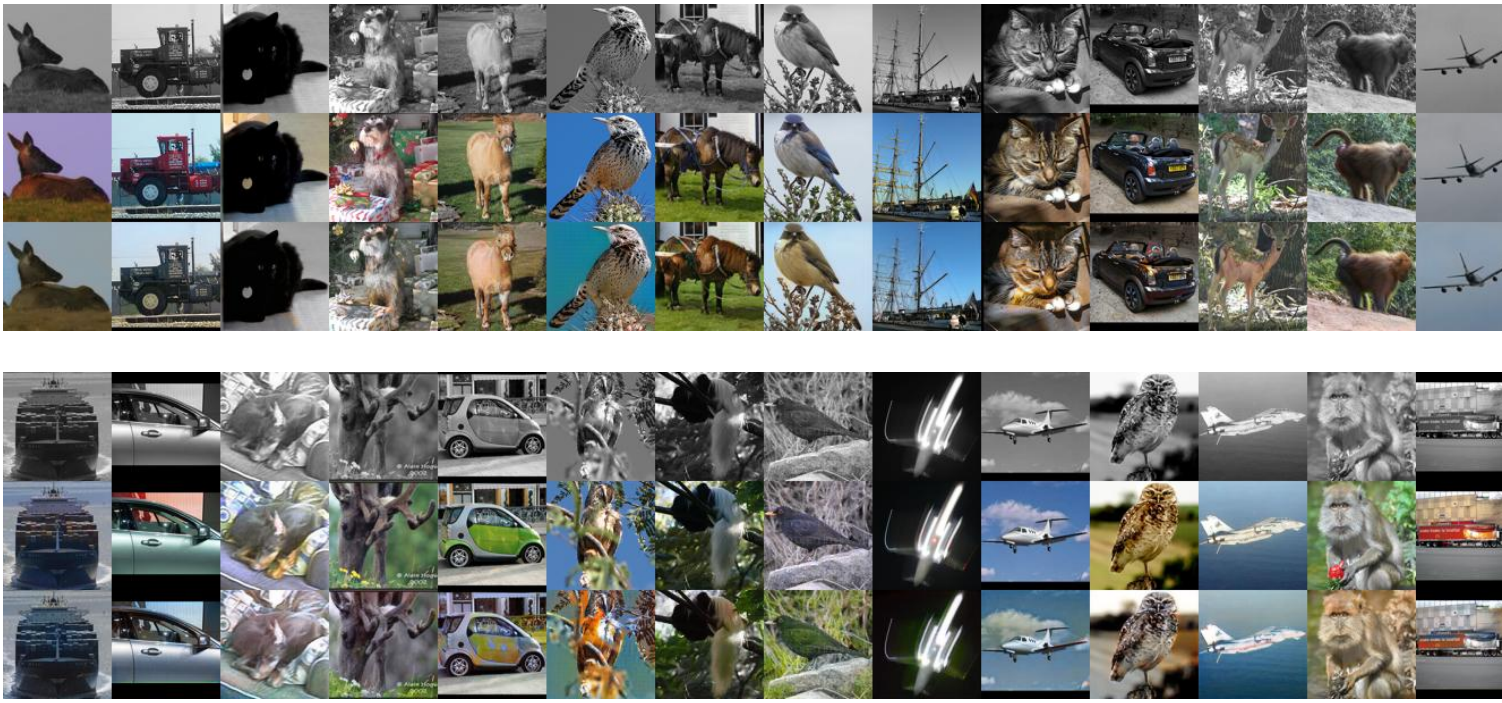
Furthermore, during training the popular data augmentation method implemented used in works such as [35, 45] were where for every training image 4 pixels are padded on each side and a 32×32 crop is randomly taken from the resulting image which is horizontally flipped by a chance of 0.5. For testing time, only the single view of the original 32×32 image is evaluated. No validation set split was used and models were trained on the full 50000 training set split for both Cifar 10 and 100.

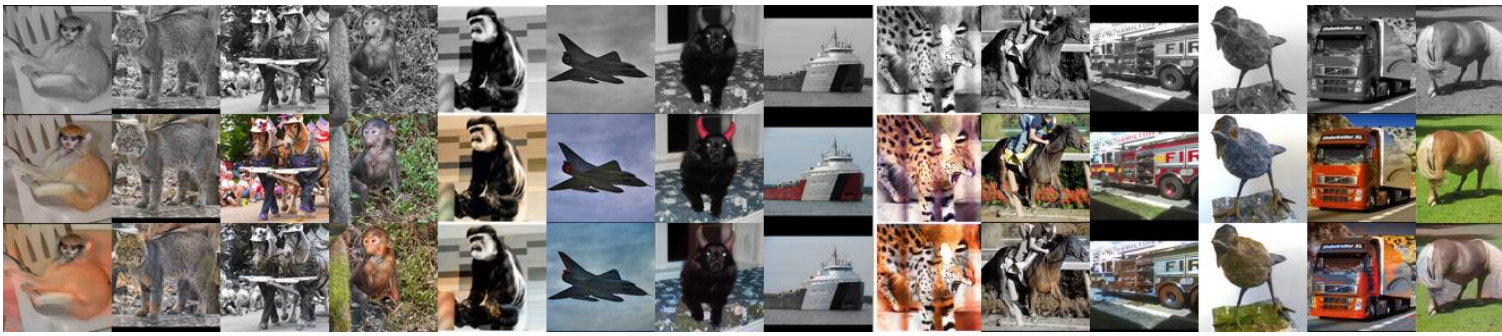
Appendix C

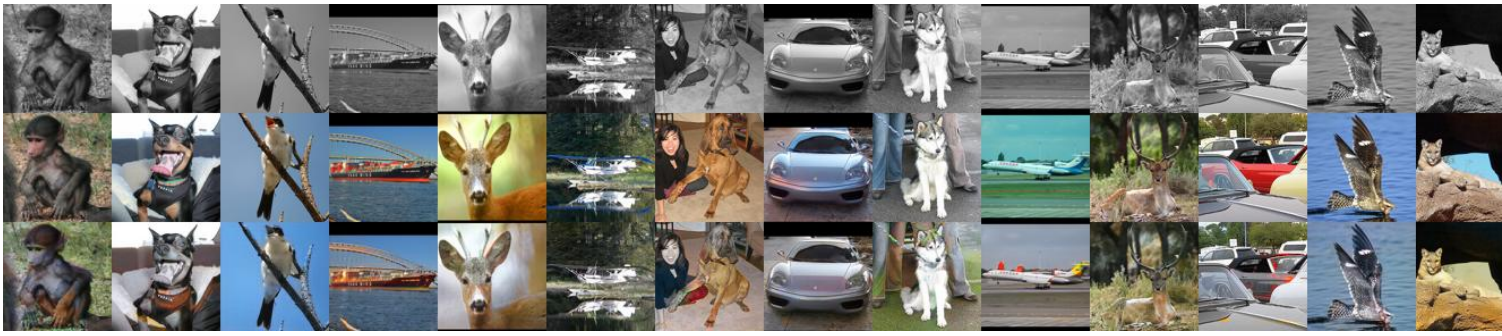
Qualitative Results

C.0.1 Samples of Generated Images from the Efficient Generative Adversarial Colourisation Network for images from an un-seen test set

For all the images shown below, the top image is the grey scale input images, the middle are the ground truth images and the bottom are the colourised images.







Appendix D

Patterns of Attention Distribution for Different Architectures and Datasets

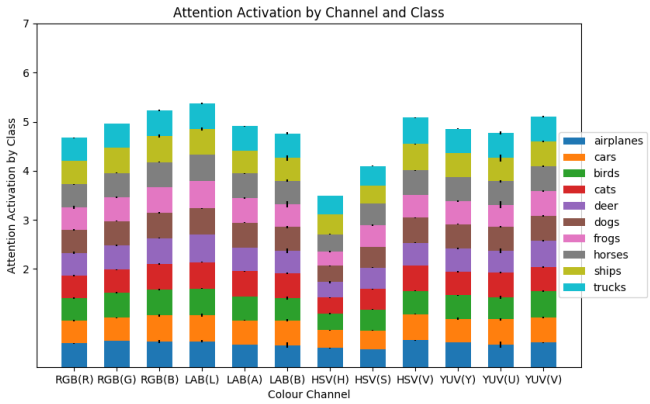


Figure D.1. Attention distribution for Densenet on Cifar10 with 12 channels

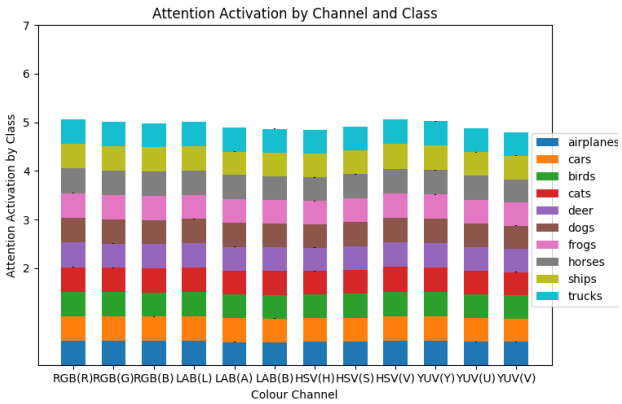


Figure D.2. Attention distribution for MobileNet on Cifar10 with 12 channels

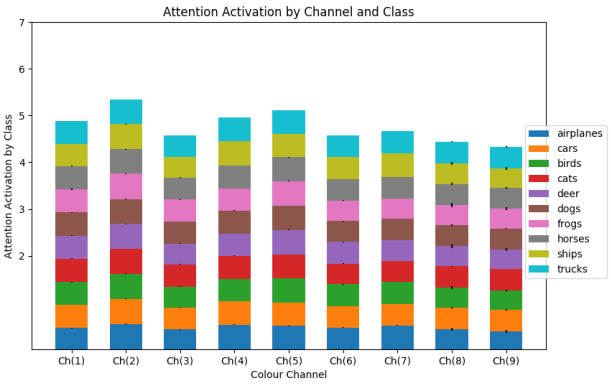


Figure D.3. Attention distribution for Densenet on Cifar10 with 9 channels

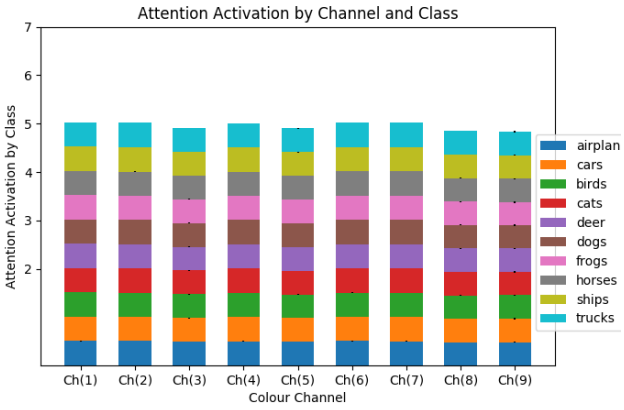


Figure D.4. Attention distribution for MobileNet on Cifar10 with 9 channels

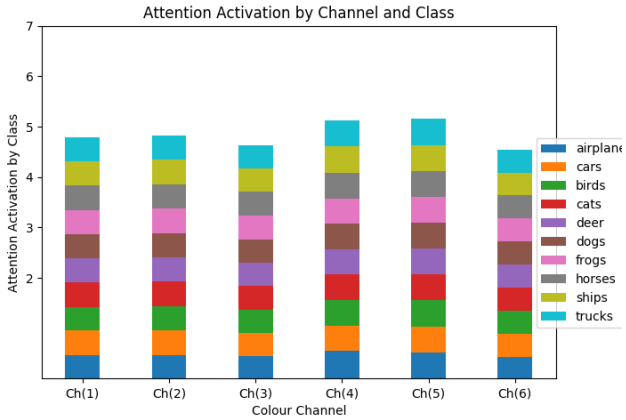


Figure D.5. Attention distribution for Densenet on Cifar10 with 6 channels

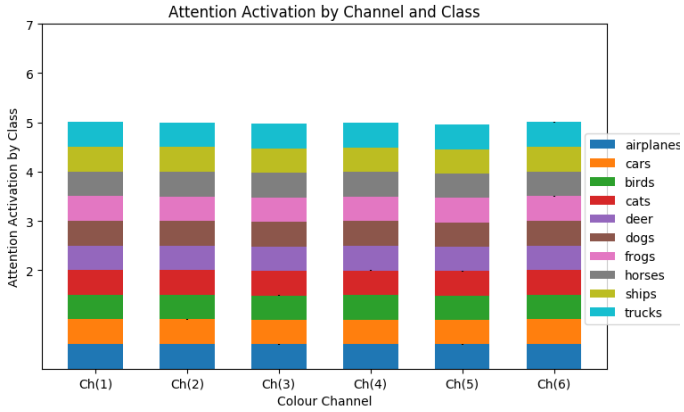


Figure D.6. Attention distribution for MobileNet on Cifar10 with 6 channels

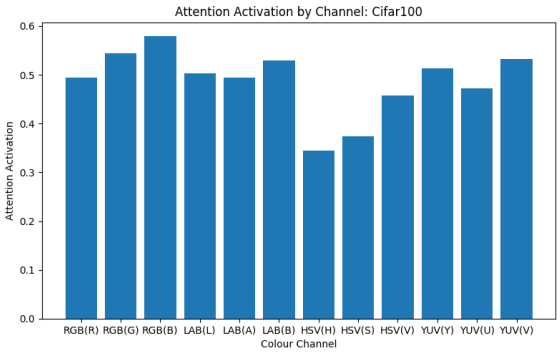


Figure D.7. Attention distribution for Densenet on Cifar100 with 12 channels

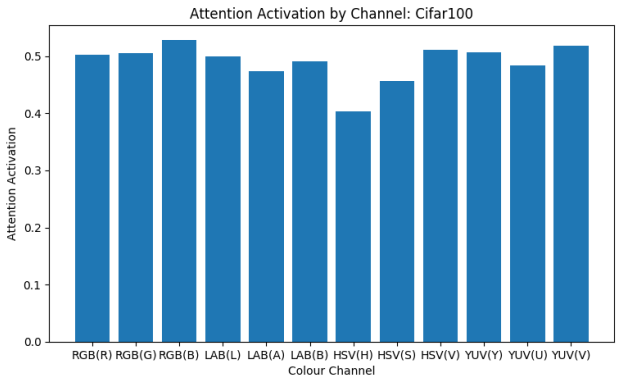


Figure D.8. Attention distribution for MobileNet on Cifar100 with 12 channels

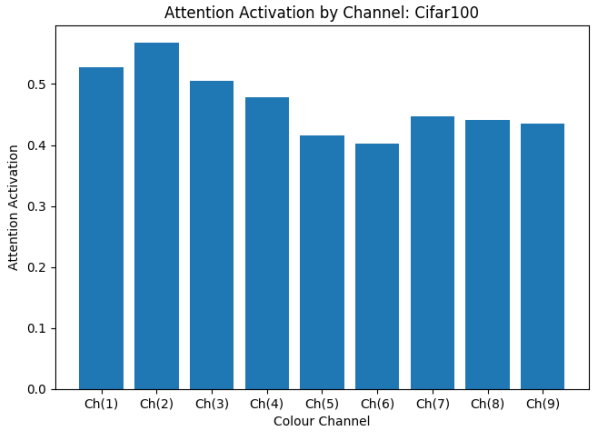


Figure D.9. Attention distribution for Densenet on Cifar100 with 9 channels

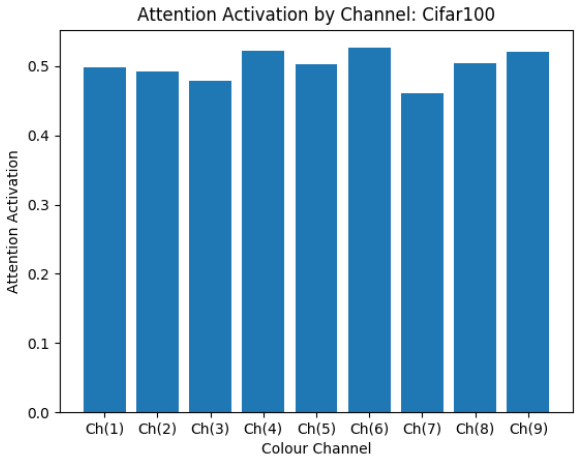


Figure D.10. Attention distribution for MobileNet on Cifar100 with 9 channels

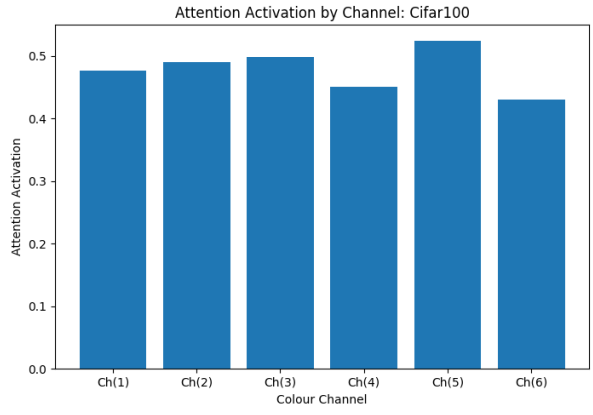


Figure D.11. Attention distribution for Densenet on Cifar100 with 6 channels

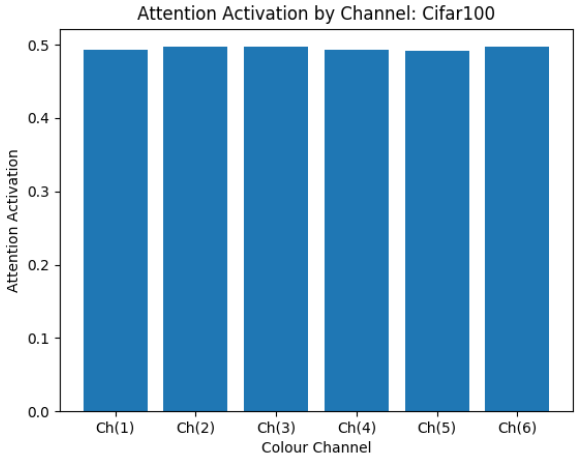


Figure D.12. Attention distribution for MobileNet on Cifar100 with 6 channels