Abstract

This project focuses on developing autoregressive models for addressing two problems using PixelCNN: denoising and colourisation. Denoising is about removing noise from images, and the project mainly focuses on Gaussian noise. For colourisation, the project introduces various ways of implementing models to colourise greyscale images with an explanation of each way: colourisation in RGB colour space, colourisation in HSL colour space, and only training the model in red and green channels.

Contents

1. Introduction

Digital images are a way of life as they play a huge role in communication, evoking emotions and capturing special moments; thus, some images could be valuable to their owners. However, the integrity of these images can be alerted, or they were vintage images, which indicates that the quality of these images is reduced due to blurring, noise, and colour loss. This reduction in image quality is called degraded images. As the era advanced in technology continuously and rapidly, the demand for images increased, which could lead to more degraded images. Therefore, developing techniques to overcome this problem has become essential.

Noise is undesirable and unwanted random variation in image brightness. It adversely impacts digital image quality, clarity, and interpretability due to incorrect capturing, processing, or transmitting. Moreover, loss of colour details often occurs in aged images because of the limitations of photographic technology at that time or intentional choice. For this reason, denoising to remove noise from noisy images and colourisation for colourising images with less colour information are necessary approaches to restore these images and keep their integrity.

Traditional image processing techniques for denoising and colourisation have been essential for enhancing image quality, which depends on manually designed algorithms based on mathematical rules. However, these degraded images could be complex data with high dimensionality, a large dataset which it is time-consuming for traditional images, or limited adaptability as the image processing relies on pre-defined algorithms and models, meaning it often displays poor performance on unseen data; hence, the requirement of an expert to create these algorithms can be crucial [1]. Given these limitations, deep learning models counter these shortcomings as they can effectively learn and adapt from large sets of data with high dimensionality and yet accomplish significant improvements on test data. Therefore, deep learning can be noticeably better in general takes for denoising and colourisation [1].

The project aims to implement autoregressive models for denoising and colourisation challenges using deep learning. To meet this aim, the pixel convolutional neural network (PixelCNN) offers a compelling case study for illustrating the effectiveness of autoregressive models in retrieving degraded images. The project's primary objective is to develop a deep understanding of PixelCNN architecture, which utilises various techniques to restore noisy images using denoising methods and accomplish accurate colours for greyscale images.

Chapter 2, related work, discusses generative models in denoising and colourisation tasks. In Chapter 3, the report will discuss the background of PixelCNN colour space, while Chapter 4 will focus on data augmentation, building a suitable neural network architecture to address both problems and training and sampling the developed networks. In Chapter 5, the report talks about the experiment, which illustrates datasets, the generated samples, and the performance evaluation of all trained models. The sixth chapter discusses the PixelCNN and its effectiveness in colourisation and denoising, as well as its limitations, and the last chapter is the conclusion.

2. Related Work

Image restoration has been one of the most fundamental tasks in computer vision in the past decade [2], [3]. It shifted from shallow networks and leaning to deep learning and neural networks as it could deal better with various takes in image restoration and outperform traditional methods [3] to reconstruct high-quality images from its alerted representations, which could be degraded due to motion of an object, limitation in resolutions, lighting, and noise caused by transmission [2], [4], [5]. Deep learning performance has been capable of solving advanced challenges in computer vision over the past few years; hence, researchers have been dedicated to developing learning schemes and analytic models that restore degraded images effectively, such as image denoising, deblurring, colourisation, and single image super-resolution [2], [3].

Denoising in general

Denoising is a technique of eliminating noise from noisy images to obtain clean images x from noisy observation y, which is given $y = x + v$, where v is the additive noise, which can be salt and pepper, Gaussian, or passion noise [5], [6], [7]. Denoising application has a huge role in medical images where noise arises from imaging equipment, astronomy due to resolution limitation, and forensic science for bad quality images [6]. Moreover, Saxena and Kourav (2014) [6] note that each problem should have a specific approach; for example, dealing with medical images is different from satellite images with the assumption that the type of noise in these images is known beforehand as it could be Gaussian, salt and pepper, or uniform. Knowing the correct type of noise is essential to construct the correct denoising technique. A convolutional neural network (CNN) is a deep learning network that has the benefit of optimising the weights of kernels continuously in training [7]. Lui et al. (2018) [7] introduce a CNN structure for denoising RGB images consisting of input, convolutional, prediction, and validation layers. For input layers, the pixel values, which usually range from 0 to 255, should be normalised between 0 and 1, as this normalisation will make the computation easier. They set the convolution later with an 11 x 11 self-adaptive kernel size to generate better images in the denoising process. The final layer ensures that pixel intensities are in the correct range, setting the values that are less than 0 to zero and more than 1 to one. They use mean square error (MSE) to calculate the error between the ground truth and the output, which will be optimised by the stochastic gradient descent (SGD) algorithm. Moreover, Zhang et al. (2017) [5] propose a sophisticated method that shifted from traditional denoising approaches. The proposed method is denoising convolutional neural networks (DnCNN), which don't output the clean image but rather the estimated noise pattern. Additionally, they explain that the inclusion of batch normalisation and residual learning significantly increases performance accuracy and training stability. This approach allows the neural network to learn the noise patterns in the image, making it efficient for various noise types. The project method in implementing denoising techniques differs from the last techniques as it aims to train and learn the model to map the relationship between noisy and clean images to output clean images using different neural network architectures, where the input images are normalised between 0 and 1.

Colourisation in general

Colourisation is a process of restoring colour information for black and white images to enhance the visuality of images [8], [9], [10]. Sometimes, RGB colour space is converted to lower dimensions such as LAB and YUV, where the prediction is only for two channels instead of 3 channels as in RGB colour space, which leads to complexity reduction [9], [10]. According to Zenger and Grgic (2020) [9], colourisation is often done using scribble-based colourisation, example-based colourisation, and deep learning colourisation. Scribble-based colourisation is based on the idea that pixels with the same intensity should have the same colour, but this method requires a significant effort of experience and time [8], [9]. The second mentioned approach is example-based colourisation; the idea is to transfer colour from a source image to a greyscale image, which is usually based on correspondences in features, textures, and segmentation, where finding the matching part between them is done by comparing the luminance values or texture patterns [9]. However, the quality of the obtained image depends on the reference image quality, and some reference image segmentation does not match the target (greyscale images) [9]. For deep learning methods, Nazeri et al. (2018) [10] introduce a deep learning colourisation method using generative adversarial networks (GANs), which consist of two small CNNs called generator and discriminator. The generator aims to map the greyscale images to the colour ones in a way that mimics the appearance of the original images, with the goal of generating indistinguishable images from the actual data. In contrast, the discriminator's job is to evaluate the generated images with an actual image from the dataset and classify them to identify whether the colour is actual. The generator improves its accuracy based on discriminator feedback. This developed network by Nazeri et al. (2018) [10] uses LAB colour space instead of RGB as the ground truth and grey image as the input. The project applies HSL colour space alongside RGB colour space and RG instead of LAB or YUV, which the report discusses in section 3.2 with the techniques of using both in colourisation in Chapter 4.

Autoregressive models: PixelCNN and PixelRNN

Autoregressive models have become famous because of their simplicity and stability, making them efficiently computable in image generation tasks [4], [11], [15]; PixelRNN and PixelCNN are examples of autoregressive models where the predicted pixel is based on the previous pixels [11], [12]. Pixel recurrent neural network (PixelRNN) scans the image pixel by pixel and row by predicting the conditional distribution of each pixel based on previously scanned pixels, which is based on RNNs. Van den Oord et al. (2016) [12] introduce PixelRNN with an innovative architecture that consists of two-dimensional recurrent layers, long short-term layers (LSTM) and diagonal bidirectional LSTM (BiLSTM). Row LSTM layer processes the images starting from the top to the bottom row by row in the sequential matter, which enables the model to get information of the current pixel based on all the prior pixels, i.e. all pixels above the target pixel and all the pixel to its left in the same row are included in the calculation but not the pixels to the right or below, which gives a triangle receptive field. The diagonal BiLSTM layer diagonally processes the image to capture pixel dependencies in two-dimensional, which captures the previous and future pixels. The model learns from the diagonal pixels by applying the skewing operation, which rearranges the diagonal pixels in a vertical manner [12]. One

difference between PixelCNN and PixelRNN is that PixelCNN utilises multiple convolutional layers to ensure that the receptive is based on spatial convolution and uses masks to ensure that PixelCNN neglects the posterior pixels. On the other hand, the PixelRNN receptive field increases dynamically to include all the prior pixels. This means that PixelRNN is processed sequentially and PixelCNN is processed parallelly during training only, but in sampling, both PixelCNN and PixelRNN generate images sequentially [12]. The project focuses on developing a standard PixelCNN that is inspired by this paper.

Colourisation with PixelCNN its variants

PixelCNN interprets RGB images sequentially, where the blue channel is conditioned on the red and the green channels, and the green channel is conditioned on the red channel; additionally, the networks output pixel values that ranges between 0 and 255 [11], [14], [15]. However, this standard PixelCNN is slow in sampling because the generation must be pixel by pixel, and it does not capture the complexity of natural images [13], [15]. Therefore, many researchers have improved PixelCNN to have many variants that incorporate enhancements to its architecture. Conditional PixelCNN is one of these variants, which models the same probability as PixelCNN with additional information, which allows the models to produce images applicable to certain conditions [14]. Kolesnikov and Lampert (2017) [15] introduce a type of conditional PixelCNN that employs auxiliary variables for natural image colourisation. They propose two generative novels to overcome these shortcomings by including an auxiliary variable to the joint distribution, called Grayscale PixelCNN and Pyramid PixelCNN. Grayscale PixelCNN inputs a standard greyscale image represented in 8-bit and a simplified 4-bit greyscale image, where both inputs are produced from the original data. This PixelCNN captures the global structure of images as it focuses on low-level details instead of high-level details; thus, it efficiently overcomes the problem of capturing the complex structures of natural images. While Grayscale PixelCNN focuses on the general structure of images, Pyramid PixelCNN aims at enhancing the generation of high-resolution images by processing hierarchical models, which start with lower-resolution images and then upsampling low-resolution images to a higher resolution. Pyramid PixelCNN achieves faster sampling because it uses low-resolution images to generate high-resolution images instead of generating the image pixel by pixel. Another technique in colourisation is introduced by Guadarrama et al. (2017) [13], which combines conditional PixelCNN and CNN to develop a colourisation technique for producing high-quality coloured images. Conditional PixelCNN is used to generate coloured low-resolution images from greyscale images; working with low-resolution images makes the model learn to assign colour information in a complexity less than high-resolution images. They train the model to generate 28×28, which is sufficiently quick, in which they produce appropriate colour layers for these reduced-resolution images. The next step is the refinement step, which is applying the image-to-image translation to the CNN network for upsampling low-resolution images with high-resolution images. This CNN takes the original greyscale image and the low-resolution output of the conditional PixelCNN network as inputs and then includes the essential depth and detail to transform the image to a larger resolution.

The project implements PixelCNN in colourisation tasks using various methods that differ from the abovementioned works. These methods use the standard PixelCNN without any variant to solve denoising and colourisation tasks, where it uses PixelCNN to process images with colour as the ground truth and CNN for greyscale images. After an extensive examination of the existing literature, no previous work for PixelCNN has been done in denoising. The project implements denoising tasks similar to the general structure of the neural network used in colourisation.

3. Background

The first section of this chapter provides a comprehensive overview of PixelCNN, discussing the autoregressive property, its relationship to CNN, masking, loss function, SoftMax, and data augmentation. The second section of this chapter explains the meaning of colour space with some illustrations of RGB and HSL colour spaces.

3.1 Theoretical PixelCNN

PixelCNN represents a significant advancement in research on generative models, which was first introduced with PixelRNN in 2016 by Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu in their paper "Pixel Recurrent Neural Networks" [12]. PixelCNN is a type of CNN characterised by autoregressive models, where each pixel depends on all the previous pixels [11], [12]. Since its establishment, PixelCNN has motivated many researchers to utilise and apply it in many areas, such as super-resolution, colourisation, deblurring, and denoising, with different variants, for example, conditional PixelCNN, Gated PixelCNN, and PixelCNN++ [14], [15].

3.1.1 PixelCNN

The autoregressive approach models the joint distribution to generate detailed and coherent images pixel by pixel. It is given mathematically as the following:

$$P(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

where it represents the probability of x_i (i^{th} pixel) given all prior pixels $(x_1, \dots, x_{i-1})$, ordered row by row and pixel by pixel [12] as shown in Figure 1 (Left).

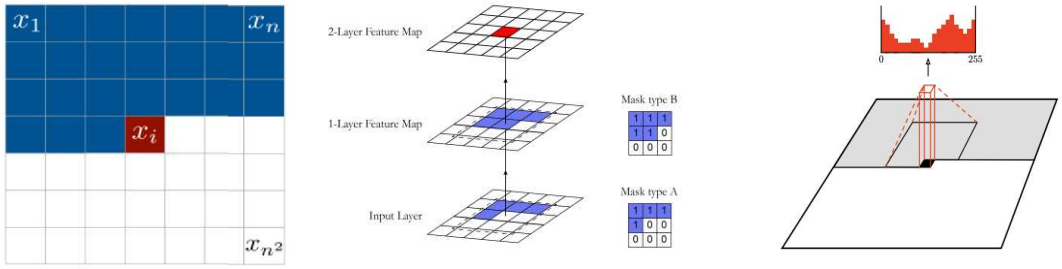


Figure 1: **(Left)** represents the joint probability of PixelCNN of the target pixel dependent on the prior pixels, the image is taken from [12]. **(Middle)** shows two types of masks, where mask type A is for the input layer and type B is for the other layers, the image is taken from [18]. **(Right)** the conditional distribution over 256 classes for each pixel, the image is taken from [14].

CNN consists of an input, convolutional, pooling, and fully connected layers [17]. Since PixelCNN is a type of CNN, PixelCNN should have an input layer and convolutional layers but not pooling layers [12]. These convolutional layers In PixelCNN should have kernels, activation maps, strides, padding, activation functions, and masking kernels only for PixelCNN, which will be discussed in section 3.1.2.

Kernels in conventional layers are small learnable metrics that convolve the entire input images to produce an activation map [16], [17], which are usually randomly initialised at the start. Activation maps are the produced output from convolving kernels through the entire image, as they enable the extraction of features from images, which play a role in updating parameters during training. Stride is the number of pixels the kernel moves across to get to a new position in the image; for example, one stride means the kernel moves one pixel at a time. Padding is adding extra pixels around the border of an image, which is usually done to preserve the spatial dimensions and control the output size [16], [17]. After the convolution operation, an activation function, often ReLU or leaky ReLU, is applied to the activation map, which introduces non-linearity that allows the network to learn complex patterns [12], [17].

3.1.2 Masking

Masking is applied to PixelCNN but not standard CNN because PixelCNN is based on autoregressive models, where only the previous pixels are included in the calculation. Masked kernels ensure the autoregressive property, which restricts the future pixels that have not been seen to enter the calculation, i.e. setting the prior pixel with one and the future pixel with 0. In PixelCNN, there are two types of masked kernels: mask A and mask B [11], [12]. The difference is mask A excludes the current pixel, considering only the previous generated pixel, and it is only used in the first convolutional layer, while mask B includes the current pixel, which the pixel is conditioned on itself and is used on all the layers expect the first one [11], [12], as shown in Figure 1 (middle).

3.1.3 Loss Function and SoftMax

Each pixel in 8-bit images often has 256 discrete possible values ranging from 0 to 255, in which 0 represents black and 255 represents white. PixelCNN outputs the probability distribution across all these 256 classes for each pixel [11], [12], as Figure 1 (right) visualises; hence, PixelCNN applies SoftMax cross-entropy.

The loss function is a fundamental task in machine learning and deep learning that quantifies how well models perform their task by comparing the predicted images to the actual images. It usually aims to minimise the loss so that the models can learn to adjust their parameters, which are weights in the convolved kernels, to improve performance accuracy and prediction. Each cost function is designed for different machine learning tasks, as regression tasks typically use mean square error (MSE). Meanwhile, classification problems could use cross-entropy [19]. The model of PixelCNN outputs 256 classes for each pixel, meaning that the model performs multi-classification for each pixel. Therefore, it applies maximum log-likelihood for model evaluation [12], equivalent to cross-entropy loss [13]. This minimisation process is usually fulfilled through an optimisation algorithm, and backpropagation such

as Adam and SGD, and the project uses the former. The cross-entropy loss function for multi-classes is given by:

$$loss(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Where y_i is the actual pixel value, and $\hat{y}_i$ is the predicted pixel value [19].

The backpropagation algorithm is a crucial step in training to minimise the loss function. According to Ketkar and Moolayil (2021) [19], the algorithm often has two main phases: forward pass and backward pass. In the forward pass, the network forwards the input data layer by layer until it reaches the final layer; the process computes the activation function (ReLU) between each layer. Then, the network computes the loss between the actual data and the predicted data using a loss function (cross-entropy). The second step is the backward pass; it starts from the output layer and calculates the gradient descent with respect to each weight using the chain rule, with the goal of updating these weights in the opposite direction of gradient descent with a small step called learning rate [19], which is the minimisation process. Backpropagation repeats for a desirable or suitable number of epochs, where an epoch means that the network passes forward and backwards through the entire training dataset only once. Each iteration in one epoch has a number of training examples, called batch size, which can be defined as the number of images processed before updating the weights; large batch size means less iteration per epoch and less weight update, and vice versa. The batch size is recommended to be in the power of 2, i.e. 16, 32, and so on. It can be determined based on the available memory in the CPU or GPU; the larger the batch size, the better, as it speeds up the training process [19].

On the other hand, SoftMax is a crucial function in deep and machine learning that converts the logits into probabilities and ensures that all classes sum to 1, i.e. it converts all 256 classes of each pixel in PixelCNN to the probability that sums to one[11], [12], [14] and it often paired with cross-entropy [20].This process allows the model to choose a probability for each pixel for image generation. SoftMax is given as the following:

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j^C exp(x_j)} \quad [20]$$

The equation takes the exponential of i^{th} class logit and divides it by the sum exponential of all logit in 256 class. During image generation, SoftMax is applied to each pixel independently with multinomial distribution to select a class from 256 classes, where the classes with high probabilities are more likely to be selected [19], [20]; this could make the pixels, in the sampling process, have different value each time. Multinomial distribution is a generalisation of binomial distribution that can result in more possible outcomes for a certain number of trials [19]; the trial in sampling is set to 1 to choose one outcome[18].

3.1.4 Data Augmentation

Dataset augmentation is essential to enhance model robustness and better generalisation. The project focuses on data augmentation for adding noise to the clean images to train the model for denoising, conversion of coloured images to greyscale images for learning colourisation, and data resizing if essential. As mentioned in Chapter 2, noise addition is given by $y = x + v$ [5], where the noisy image is y, x is the clean image, and v is the additive noise. Gaussian noise is the additive noise that is used in this project when dealing with denoising takes. Greyscale has equal intensities of all colours in RGB colour space. There are three common ways to convert RGB colour to greyscale, which are the lightness method, the average method, and the luminosity method [21]. The project focuses only on the average method and the luminosity method. The average method takes the mean of all three channels in RGB (red, green, and blue), given by the following equation: $Grey = (R + B + G)/3$, while the luminosity method calculated a weighted average of values that are sensitive to the human eyes, rather than taking the mean, its weights are given by $Gray = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$ [21]. Figure 2 shows the application of these two methods on image **A**, where **B** is converted using weighted values, and C takes the mean value.

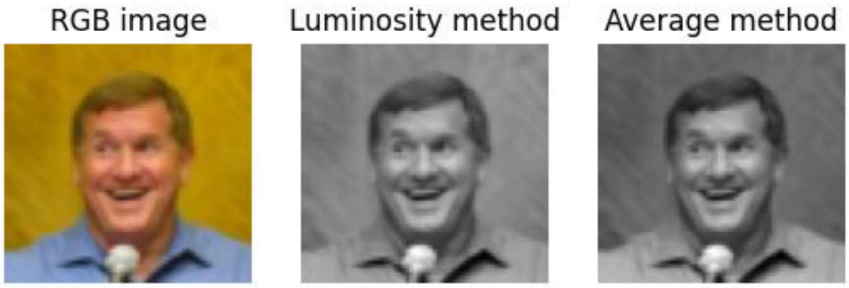


Figure 2: (**A**) (**B**) (**C**)

3.2 Colour space

Colour is a reaction of the human brain to the visual perception of light, which is determined by wavelength, intensity, and context. It can be described by hue, lightness, colourfulness, saturation, and chroma [22], [24]. Colour space models are a representation of mathematical models that describe how colours are identified, for example, RGB, HSV, LAB, and HSL [23]. Each colour type in colour space can serve a different purpose because some colours could be complex compared to the other space colours, or some applications require specific colour space [21]. Colour space is categorised as either device-dependent colour space such as RGB, HSV and HSL, or device-independent colour space, for instance, CIELAB [22], [23]. Device dependency means that some devices might render the same value of the same colour spaces differently.

Colour space is crucial in computer vision as it plays a huge role in image processing. Velastegui et al. (2021) [23] emphasise several points to highlight the importance of colour space in computer vision.

One of the mentioned points is the need to choose a proper colour space for specific tasks in computer vision that could result in stable optimal outcomes; for example, the selection of appropriate colour space in image classification could improve and enhance the accuracy performance significantly, especially if the artificial neural network is trained in a colour space other than RGB as RGB colour space consider complex compared to the other colours.

RGB colour space

The red, green, and blue (RGB) colour space is widely regarded as one of the best device-dependent colour spaces, with its usage popularity in almost all computer systems [22], [24]. It is based on tri-chromatic theory, with the advantage of easy implementation despite its non-linearity with visual perception [22]. RGB is represented by a tuple (R, G, B) with each component specifying its intensity; when RGB is black, it means all the three primary colours are absent $(0, 0, 0)$, but when all the colours are full intensity, the colour is white $(255, 255, 255)$.

HSL colour space

HSL is short for Hue, Saturation, and Lightness/Luminance. It is frequently used in many areas of computer vision, such as robot vision to identify entities and medical imaging. According to Saravanan et al. (2016) [24], the meaning of each component in HSL colour space is the following:

- Hue (H) is considered a pure colour that ranges from 0 to 360 degrees.
- Saturation (S) measures the vividness and purity of the colour, ranging from 0% to 100%. 0% saturation means the colour is grey, while 100% means the colour is in its most intense and vivid form.
- Lightness or Luminance (L) is the brightness of the colour, which represents the same concept of grey images in RGB colour space. A lightness of 0% means black and 100% means white; therefore, lightness in HSL has similar properties as greyscale in RGB.

Lightness in HSL is computed by taking the mean of the maximum and the minimum value in RGB colour $L = (c_{max} + c\_min)/3$ [24]. HSL colour space can be used in colourisation to reduce the complexity rather than using RGB, where the model learns the relationship between the lightness (L) and hue and saturation (HS) during training, enabling the model to predict HS from L. The project implemented two functions that convert RGB to HSL and HSL to RGB based on the math provided by Saravanan et al. (2016) [24]. This conversion could make some pixel's value outrange or undefined numbers, so it is essential to make sure all the values in the image are between 0 and 1 and is a number.

4. Methodology

This chapter gives a brief overview of the PyTorch framework and introduces the techniques for colourisation and denoising that have been implemented. It presents the methodology for the neural network architecture, data preparation, different techniques, training process, and sampling. These techniques are developed in the proposed network architecture, which are trained and sampled using a graphic processing unit (GPU).

PyTorch framework

PyTorch is an open-source framework built by Facebook using Python. It is used in machine learning and deep learning and supports GPU acceleration. It includes an auto-grad system that simplifies the backpropagation process [25]. It uses a fundamental component called tensors, which are a multi-dimensional array that has GPU support and interoperability with the NumPy library, which makes it easier to switch back and forth [25]. This design makes it easier to develop complex models and train them with a readable syntax.

4.1 Neural Network Architecture

The network needs to handle two inputs, which allows it to learn the relationships between ground truth as the first input and the corrupted images with colour loss as the second input. The idea is to build an architecture that employs three networks where PixelCNN processes the first input, where the model aims to learn how to map the degraded images to the PixelCNN network. Meanwhile, CNN processes the second input, which can be noisy or grey images, and lastly, CNN network processes the concatenation of both PixelCNN and CNN. ReLU is the applied activation function between the layers in all three networks with batch normalisation. The inputs are normalised between 0 and 1 as it eases the computation and stability, as explained in Chapter 2.

The role of PixelCNN in this architecture is to learn the process of generating the target images, i.e. generating clean images from noisy images and coloured images from greyscale images. The reason behind using PixelCNN is to learn the joint distribution of pixels in original images, such as clean and coloured images, to capture the receptive field in an autoregressive way that achieves the aim of the project in generating images in an autoregressive way. This model captures the complex dependencies between pixels, meaning it learns the colour distribution with various objects when dealing with colourisation tasks or learns the textures and structure of clean images without noise for denoising tasks. As mentioned before, the last layer of this model must output 256 discrete values for each pixel.

CNN's role is to process and extract useful information, features, and patterns from the corrupted (noisy) or greyscale images since CNN networks are known for their powerful extraction of features [16]. As both networks (PixelCNN and CNN) are concatenated before being processed in the final network, both networks must be compatible in dimensions and the number of output classes for each pixel. To achieve this compatibility, pooling layers in CNN should be avoided as they decrease the spatial dimensions and the size of the outputted image; additionally, CNN must output 256 classes

similar to PixelCNN, but they can have different kernel sizes, padding and the number of hidden layers as long as they have similar output classes.

The next step is concatenating the outputs from PixelCNN and CNN, resulting in an activation map with 512 discrete values over each pixel. This output is considered an input for the special CNN which processes the data using 1×1 kernels and stride 1. Convolutional layers of size 1×1 can have many advantages over other kernels' sizes. A kernel with size 1×1 can mix features of initial inputs that allows the model to learn the important features. In other words, it learns to reconstruct clean and colour features from noisy and greyscale images for each pixel while maintaining the resolution, and it increases the ability of the model to capture non-linearity within pixels. This special CNN must output 256 classes at the output layer.

Denoising

Training the model on denoising tasks requires the preparation of a dataset in a way appropriate for the network architecture. The data loader in PyTorch should have images in pairs, where the pair contains original clean images and noisy images that are created from adding Gaussian noise to the clean images. In PyTorch, the process involves creating a Gaussian function that adds noise to a tensor representing a clean image. The project produces two models that handle different levels of noise; one processes a lower level of noise with 0.1 standard deviation and 0 mean, and the other model operates with a noise intensity of 0.3 standard deviation for addressing the higher-level noise. Denoising in this report talks only about images with one channel, namely greyscale images. Since input images are typically normalised between 0 and 1 in the preprocessing stage, and the model is constructed to output values within the range 0 and 255, matching the 256 different classes, the ground truth or target is multiplied by 255 to calculate the loss between the ground truth and the model's output using the cross-entropy loss function.

Colourisation

In PixelCNN, each channel in any colour space is conditioned on the previous channels; for example, consider RGB colour space where the B channel is conditioned on R and G, and the G channel is conditioned on R. One way to achieve this process of capturing the correct receptive field of PixelCNN, is to transform the data of 3-channels or 2-channels into a flattened single row layout, where each channel is sequentially aligned along the width dimension for each row, resembling the format in Figure 3 (A) which shows an example for RGB images of shape $[Channels: 3,\ Height: 4,\ Width: 4]$. Colourisation in training processes a pair of inputs: coloured and grey images. Thus, both must originate from the same image, and they should be compatible in shape and dimensionality, which makes it necessary to adjust the greyscale images in a way that has the same shape as coloured images. Grey images generally have one channel, where each grey pixel maps to all corresponding pixels in each channel in the coloured image; therefore, each pixel is duplicated three times or two times depending on the shape of the flattened coloured images. Figure 3 (B) visualises the structure of greyscale images in colourisation. The dataset is prepared in pairs, each of which consists of the reformatted colour images with their corresponding grey images. The neural network receives two

inputs as constructed and then outputs the predicted values of 256 classes. Since updating the models' weights requires calculating the cross-entropy loss, the flattened colour images are used as the ground truth, and it must multiply by 255, as discussed earlier.

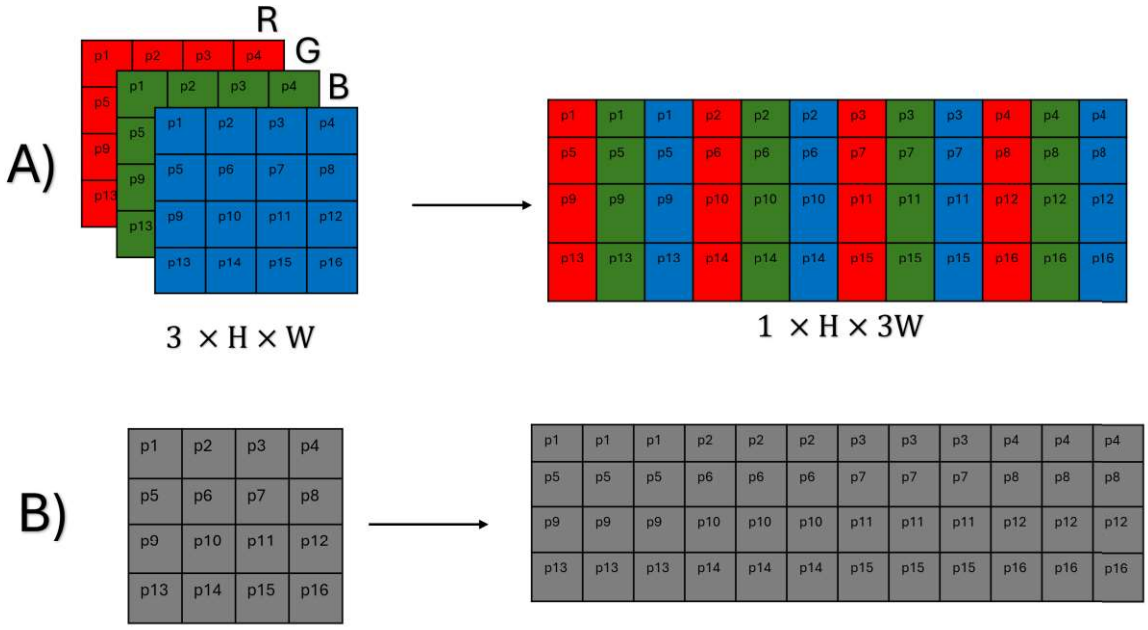


Figure 3: data preparation for colourisation, **(A)** transformation of channels in sequential order, each square represents a pixel. **(B)** shows the structure of duplicating the pixels for greyscale images.

The project developed many techniques of image colourisation, which are processing RGB images and HSL images, and the last approach is training the model on only two channels RG of RGB as the third channel (B) can be computed from red and green channels using the grey image. Another method is to adjust the neural network architecture in a way that it can learn and recognise which channel it learns on, which is to increase the number of input channels to two in CNN that process the grey images, which adds another layer to greyscale images that contain the channels number, for simplicity this process in the report is called "the additional channel". Figure 4 (right) shows RGB colour space, which has three channels: red (R) represents channel 0, green (G) represents channel 1, and blue (B) represents channel 2; meanwhile, on the left, channel 0 can represent hue (H) or red (R), and channel one can represent saturation (S) or blue (B) for HS and RG.

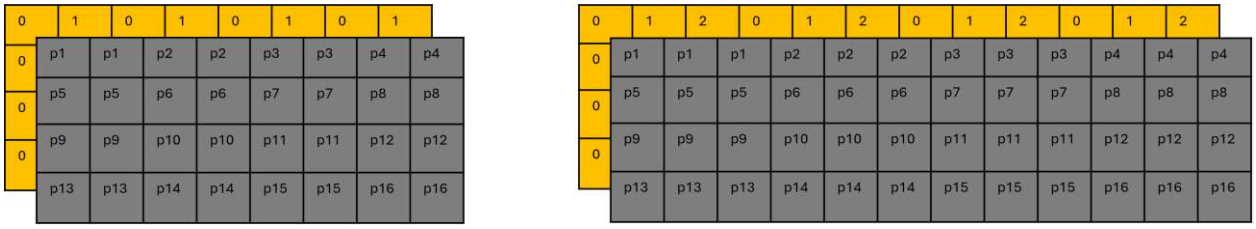


Figure 4: the general structure of adding the additional channel with greyscale images. **(Left)** represents the additional channel RG and HSL methods, **(Right)** represents the additional channel for RGB method.

For processing the RGB dataset, the greyscale images are created for the original using a grey transform in PyTorch, which converts RGB to greyscale using the luminosity method, as discussed in section 3.1.4, where the RGB image is used as the ground truth. For HSL colour space, the RGB dataset is converted to HSL in which the pair of datasets consists of lightness (L) and HS, as described in section 3.2, where hue and saturation are used as the ground truth. In the last technique, the greyscale images are created by using the average method so that the model trains only on the red and green channels, where RG is used as the ground truth. Choosing another colour space than RGB or training fewer channels reduces the complexity of training significantly, as the model only learns on two channels (HS and RG) compared to three channels in RGB space, as mentioned in Chapter 2 in the colourisation section.

4.2 Sampling

PixelCNN generates images row by row and pixel by pixel, starting from a blank image (tensor filled with zeros) that has the same size as the model trained on. Each generated pixel is then added to the blank image, which is used to generate the next pixels conditioned on it and all the previous pixels. This ensures that the generated pixels are used to generate the current pixels. Each predicted pixel uses SoftMax to convert the logits to probabilities with multinomial distribution to choose one class from all the other 256 classes. This process is repeated until all the pixels in the images are generated, which could be a clean image for denoising or a coloured image for colourisation, as shown in Figure 5 from the left to the right. For denoising, the number of generated pixels is $(H \times W)$; however, for colourisation, the model generates all the pixels in the flattened form, their total is $(H \times 3W)$ or $(H \times 2W)$. After sampling, the result from colourisation should be reshaped to the original, which is reshaping the RGB image to its original shape of 3-channels, while HSL starts with reshaping the HS into two channels and then the addition of the lightness (L) as their third channel, so it can be converted back to RGB colour space, and lastly, for RG it reshaped into two channels, and then compute the third channels as $B = 3 * Grey - (R + B)$. These samples should be converted to NumPy shape to visualise them, which is represented in this shape $[H, W, C]$.

4.3 Evaluation Metrics

The evaluation of a model measures the difference or similarity between the ground truth and the generated images [26]. There are many evaluation metrics, and each has a different purpose. Denoising and colourisation models often have three metrics to evaluate their performance, which are RMSE, PSNR, and SSIM [26], [27]. The report will focus on PSNR for evaluating denoising models and SSIM for colourisation model evaluation.

Peak signal-to-noise ratio (PSNR) is an evaluation metric that uses mean square error (MSE) [27] between the clean and noisy image in denoising. PSNR is mathematical is given by:

$PSNR = 20 \times \log_{10}(\frac{MAX}{\sqrt{MSE}})$, where MAX is the maximum value of pixels in the grey images, and MSE is given by $MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, where n is the total number of pixels in an image, y_i is the original pixel image, and $\hat{y}_i$ is the generated pixel from the denoising process [26], [27]. The larger the PSNR value, the higher the similarity is, and vice versa [26].

The structural similarity index (SSIM) metric measures the perceptual similarity between two images; it considers the change in structure, luminance, and contrast [26], [28], which could make it better than PSNR in evaluating the change in luminance and structure. It measures the mean, variance, and covariance of the image's structure, constant, and luminance. The formula is given as $SSIM(X,Y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + c_2)}{(u_x^2 + u_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$, where X, Y are the ground truth and the generated image respectively, u, σ are the mean and variance of either X or Y, σ_{xy} is covariance, and c is just a constant, where the higher the score of SSIM the higher the similarities and lower the differences [26]. It ranges between -1 and 1, where 1 means identical images, 0 means totally different, and -1 means perfect negative correlation [28].

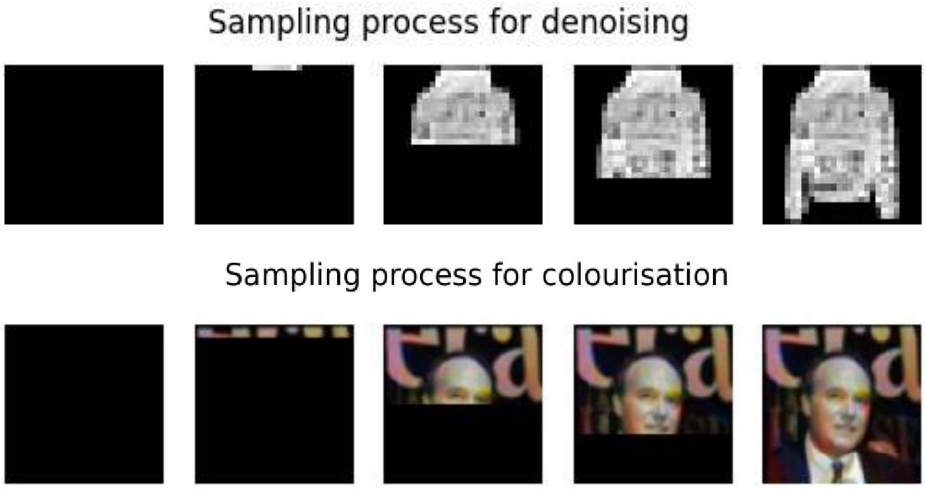


Figure 5: an example of the sampling process for denoising and colourisation.

5. Experiment

This chapter presents an experiment of the methodology, discussing the used training and testing datasets, examining experiments for training and sampling, showing the results of all used techniques, and evaluating these results.

5.1 Approach and Procedures

Denoising datasets

Two datasets containing greyscale images are used in training denoising models. The first one is the MNIST dataset, a large dataset consisting of 70,000 handwritten digit images, where 60,000 are used in training and 10,000 in testing; each image has a size of 28×28, which can be found in [29]. The other dataset is Fashion MNIST, which consists of 10 different fashion classes, and each image is 28×28, which can be found in [30]. It has the same number of images in the training and testing as MNIST. MNIST dataset is only applied to one model that learns to generate clean images using images with low-level noise. In contrast, the Fashion MNIST dataset is used in both models, which deal with a low Gaussian noise and a higher Gaussian noise.

Colourisation datasets

For colourisation, the CIFAR-10 dataset is used in the training process for one of the colourisation techniques, which is RGB colour space with the additional channel that makes the neural network recognise the channel numbers. The CIFAR-10 dataset consists of a total of 60,000 images of size 32×32 divided into ten classes, where 50,000 images are used as the training dataset and 10,000 as the training dataset, which can be found in [31]. The other five methods in colourisation utilise the LFW dataset during training. LFW is a dataset that contains the faces of more than 13,000 people and is divided into training and testing, where each image is size 250×250, which can be found in [32]. These images are resized to 64×64 because training on a lower resolution is faster than higher resolution, and also, the provided GPU acceleration is limited in processing large images. For colourisation in RGB colour space, the shape of images in the dataset, after flattening the three or two channels into a one row layout, is $[C: 1, H: 64, W: 192]$, while images in HSL colour space and RG method have the shape $[C: 1, H: 64, W: 128]$ as they have two channels, RG and HS.

Training for colourisation and denoising

In the training process, each technique has spent a different time and number of epochs, and each method has a different number of hidden layers. In PixelCNN, the mask of type A is applied to the first layer, and mask B is applied to the hidden layers with ReLU and batch normalisation between the layers. Batch normalisation could improve the overall performance of a model [5], as mentioned in Chapter 2. During training, the learning rate for denoising models is fixed; however, for colourisation, the training process starts with a high learning rate, then is reduced between the 50th and 70th epochs. This process of starting with a high learning rate leads to faster convergence in early training, and then the learning rate is reduced to help the model avoid overshooting, which is when it fails to converge.

The strategy of deciding the number of epochs usually depends on the loss function, as a steady loss for several epochs either means that the learning rate could be reduced to avoid overshooting the optimum or the model has trained enough if the learning rate is the standard default value (0.001). It is essential to refrain from overtraining any model more than it is supposed to, as it could lead to overfitting on unseen data. After training terminates, the trained model and its learned weights and parameters are saved to a file for generating the aimed images.

The denoising model dealing with 0.1 standard deviation is trained on two datasets; one generates handwritten digits, while the other generates fashion. The training process took roughly one and an hour for 20 epochs for the MNIST dataset, whereas the Fashion MNIST dataset took 35 epochs in an hour and a half. On the other hand, the model that processes more level noise was trained only on the Fashion MNIST dataset, which lasted around an hour and a half for 40 epochs. These three models have the same network depth, where PixelCNN has nine layers, including the input layer, CNN has five layers, and the special CNN has six layers. These small datasets that were used in the training denoising model allowed faster training compared to other datasets and the other models because of their image size.

For RGB colourisation without the additional channel, the model trained for 187 epochs for around 15 hours, where it has 11 layers for PixelCNN and CNN and eight layers for the special CNN, while the other RGB model with the additional channel lasted 6 hours for 100 epochs, which has seven layers for PixelCNN and CNN, and six layers for the special CNN. For HSL colour space models, HSL colour space without the additional channel was trained for 95 epochs and lasted 5 hours, and HSL with the additional channel was trained for 118 epochs, which took 6 hours. The applied layers for these two models are 9, 8, and 7 for PixelCNN, CNN, and the special CNN, respectively. For the last two models trained on RG, the training process for RG without the additional channel took 150 epochs that lasted 6 hours. It trained on 11 layers for PixelCNN and special CNN, and nine layers for CNN, while the model with additional channel trained for 168 epochs that lasted 7 hours and has 11, 9, and 7 layers for PixelCNN, CNN, and the special CNN. The decision on the number of layers or the reduction of the learning rate depends on experiments after training different models on different layers and learning rates.

5.2 Results

This section shows the results obtained from generating images using the trained models: three models are for denoising, and six are for colourisation. Each model creates three samples for one noisy or grey image, where a sample is just a possible generated image based on the probability of the network prediction. The aim is to produce a sample close enough to the original.

Denoising result

Figure 6 shows the denoising results for two models trained on MNIST and Fashion MNIST datasets. These models generate clean images from images with a Gaussian noise of 0.1 standard deviation.

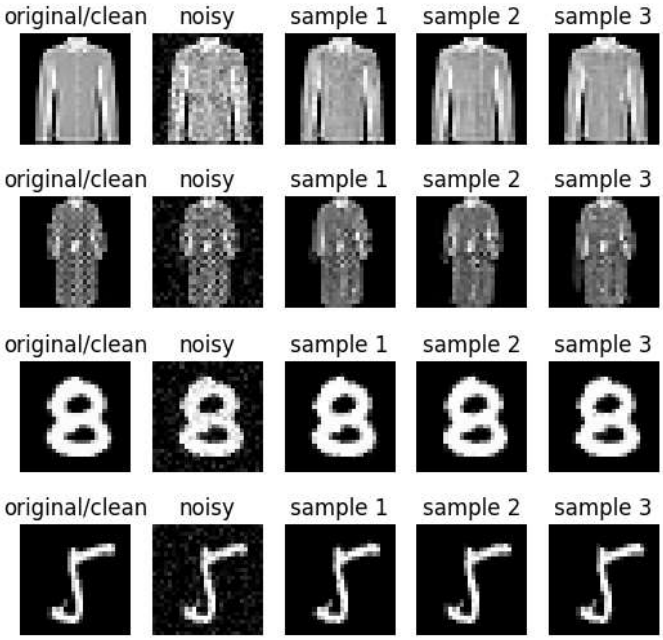


Figure 6: Denoising images with 0.1 standard deviation (low level noise).

Figure 7 presents denoised generated images from a model trained on Fashion MNIST that has a Gaussian noise of 0.3 standard deviations. The sampling time for all the samples in Figure 6 and Figure 7 took one minute.

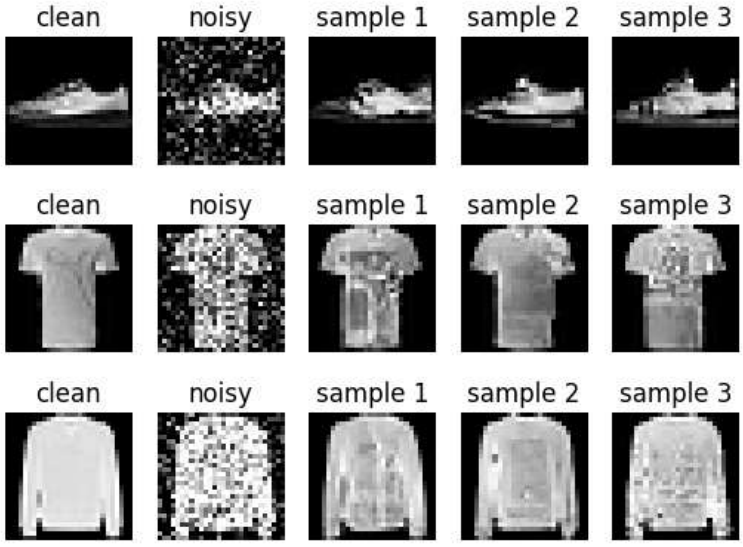


Figure 7: Denoising images with 0.3 standard deviation (higher level noise).

Colourisation results

Figure 8 shows the results of colourising the grey images, where the model is trained on RGB colour space without the additional channel. The network generates three samples (one row) in 7 minutes.

Figure 8: Samples for model trained on RGB images.

Figure 9 illustrates the results generated using the trained model on RGB (CIFAR-10) image with the additional channel. The model creates three samples in four minutes.

Figure 9: Generated coloured CIFAR-10 images.

The next two figures show the results produced by two models trained on HSL colour space, where LFW is used as a dataset in training and sampling. The generated images are converted from HSL to RGB colour space for visualisation. Figure 10 represents the converted HSL images without the additional channel in the neural network, while Figure 11 shows the converted HSL images with the additional channel. Each row took 2 minutes for sampling and conversion.

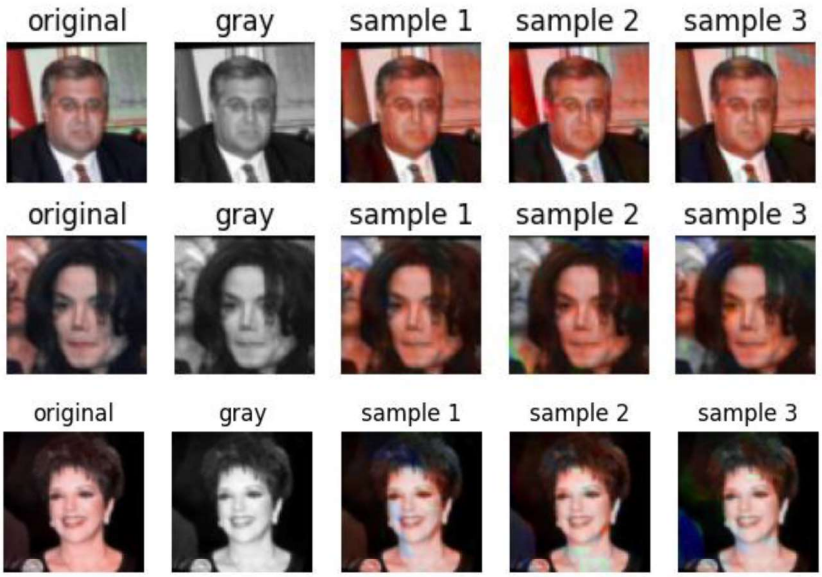


Figure 10: HSL without an additional channel.

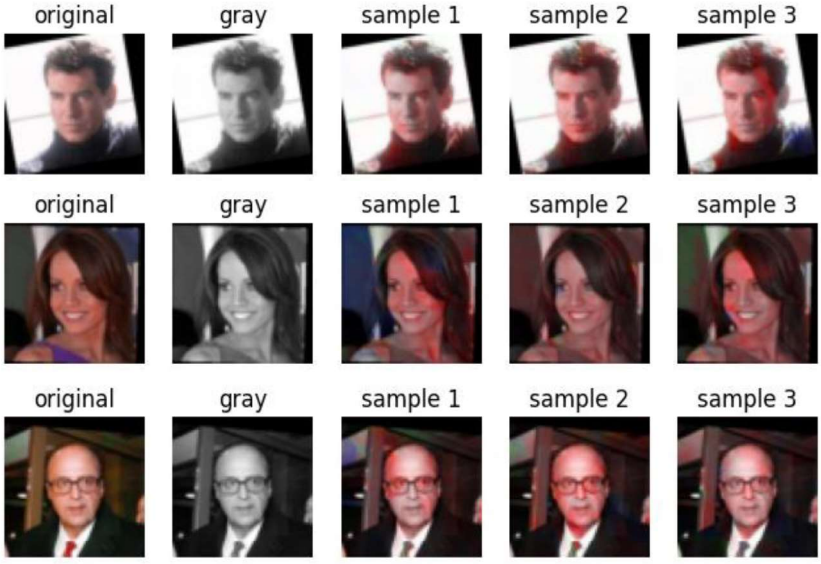


Figure 11: HSL with an additional channel.

The following two figures visualise coloured RGB images that is produced using the generated red and green channels (RG), where Figure 12 represents the generated images without the additional channel, where the second row is trained on a neural network that has more depth than the samples in the first row (i.e. increasing the number of layers in the network), to test if the result would improve. Meanwhile, Figure 13 shows the result with additional channel inclusion.

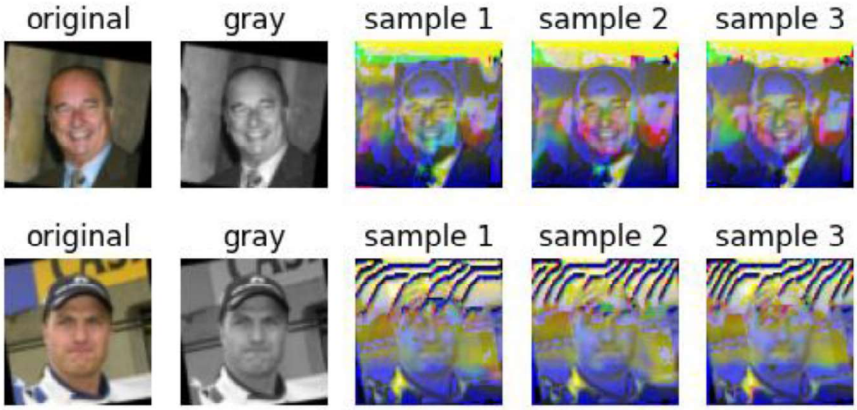


Figure 12: RG images without an additional channel.

Figure 13: RG images with an additional channel.

5.3 Evaluation

PSNR is used for evaluation the performance of denoising model and SSIM for colourisation. The following figures shows PSNR and SSIM for each generated image and sample, then Table 1 and Table 2 shows the overall performance of each model.

Figure 14 shows the evaluation of samples using the PSNR metric for denoised images that have the discussed noise intensities, where the top row in the figure has a low-level noise and the bottom a higher-level noise. When the mean square error (MSE) metric equals zero, it means that the two images are identical, as shown in the figure, where the clean image is compared to itself.

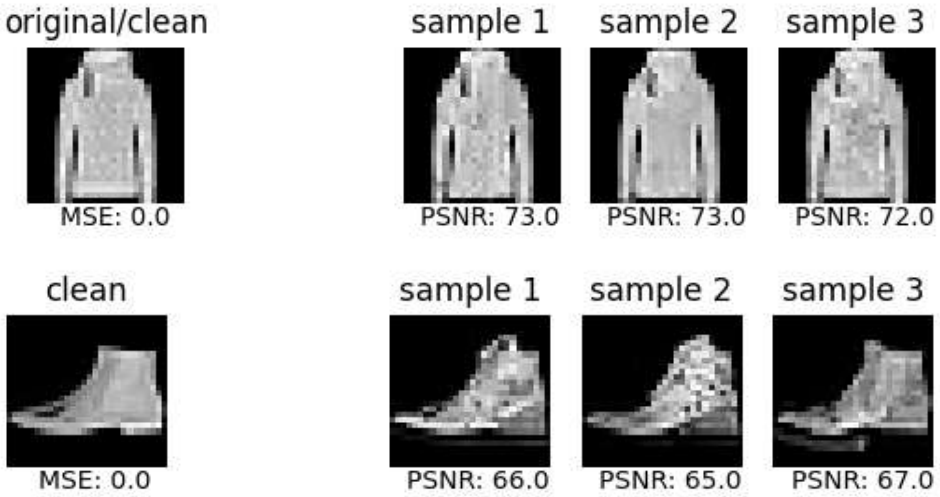


Figure 14: Applying PSNR for denoised images. (**Top**) A sample with low-level noise, and (**Bottom**) sample with higher-level noise.

The SSIM metric evaluates colourised images from greyscale images, where 1 means the image is identical, as shown in the following figures when comparing the reference image to itself. Figure 15 shows the measurement of SSIM on generated RGB samples, where the top row represents RGB colourisation and the bottom RGB colourisation with the additional channel in the neural network.

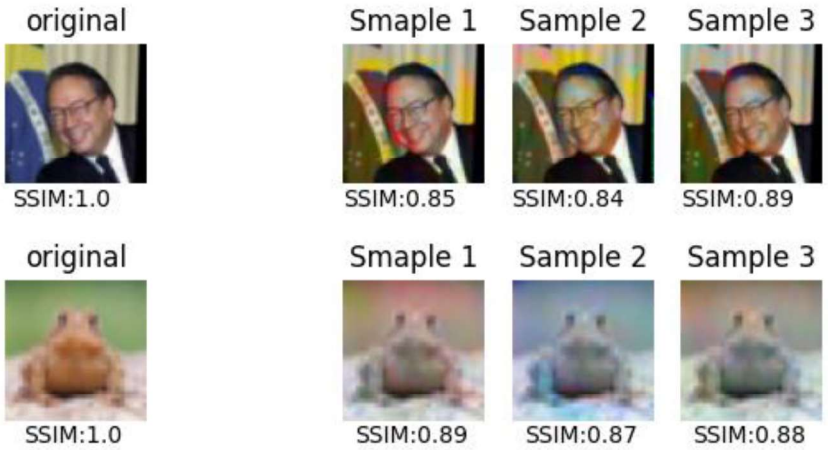


Figure 15: SSIM metric in colourisation of RGB colour space, (**top**) model without the additional channel and (**bottom**) samples with the additional channel.

Figure 16 shows the SSIM evaluation in HSL colour space, with the top samples generated by the proposed network without the additional channel and the bottom samples generated with the additional channel.

Figure 16: SSIM in samples created by models trained in HSL colour space. (**Top**) HSL without the additional channel, (**bottom**) HSL with the additional channel

The last figure (Figure 17) displays the evaluation of the SSIM metric images generated using the RG method. The top row is for RG, which excludes the additional channel, and the bottom row includes the additional channel.

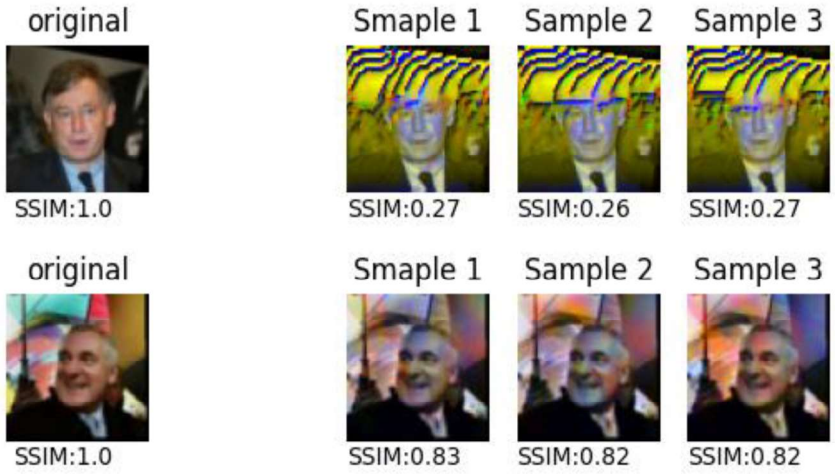


Figure 17: SSIM on samples that is generated using RG technique. (**Top**) RG without the additional channel. (**Bottom**) RG with the additional channel.

Evaluation of the overall performance of a model requires calculating the average of metric scores for a set of generated images from the same model. The following two tables explain the overall performance for the average metric score for 20 generated images of each model in denoising and colourisation. Table 1 shows the average PSNR metric scores for 20 denoised images using the three trained models, while Table 2 states the average SSIM metric for 20 colourised images using the six trained networks, as explained in sections 5.1 and 5.2.

Models	**Datasets**	**Gaussian noise intensity (standard deviation)**	**PSNR metric evaluation score**
Model 1	MNIST	0.1	76.56
Model 2	Fashion MNIST	0.1	73.73
Model 3	Fashion MNIST	0.3	65.34

Table 1: The average PSNR score for 20 generated images by the denoising models.

Models	With/Without the additional channel	Datasets	SSIM metric evaluation score
RGB	without	LWF	0.86
	with	CIFAR-10	0.88
HSL	without	LWF	0.87
	with	LWF	0.9
RG	without	LWF	0.30
	with	LWF	0.85

Table 2: The average SSIM score of over 20 images generated using the colourisation models.

6. Discussion

This chapter discusses three sections, starting with PixelCNN's effectiveness in denoising and colourisation using the proposed techniques, explaining the limitations, and ending with future work suggestions.

6.1 Result Analysis

The denoising models trained on the proposed network architecture noticeably generate high-quality images, indicating that PixelCNN can efficiently denoise noisy images with the used processing technique, as illustrated in Figure 6 and Figure 7. Model 1 in Table 1 produces a high PSNR score for denoised MNIST images of 76.56 with low-level noise, which indicates an excellent clean generated sample from a noisy image. Model 2 deals with the same noise intensity as Model 1 but uses the Fashion MNIST dataset, which produces a high score of 73.73. Model 1 generates higher quality than Model 2 because handwritten digits have simpler patterns and contrast between the digit and the background than Fashion MNIST, making Model 1 faster in training with better quality than Model 2. Regarding Model 3, it generated clean images from higher level noise that got a PSNR score of 65.34, less than Model 1 and Model 2 but still a high-quality generated clean image. Increasing noise intensity can reduce the PSNR score, but it does not necessarily mean a reduction in quality.

The colourisation models generate various colourised images that have SSIM scores ranging from excellent to poor quality. Table 2 shows all the trained models using PixelCNN and their overall performance in colourisation. The models developed using RGB colour space usually provide plausible results, as presented in Figure 8, Figure 9, and Figure 15. The average SSIM score of 20 images is 0.86 for the RGB model that is trained without the additional channel and 0.88 SSIM score for RGB with the additional channel. For HSL colour space, the model could produce an outstanding to plausible colourised image, as shown in Figure 11, Figure 10, and Figure 16, where the model without the additional channel has an SSIM score of 0.87, while the model that has the additional channel scores 0.9. Moving to colourisation in RG methods, the model without the additional channel has SSIM scores of 0.3, which generates poor-quality images in structure, luminance, and contrast. The RG model that has the additional generates plausible to good images, as shown in Figure 13 and Figure 17 (bottom), where it has an SSIM score of 0.85.

Table 2 shows that training models on colourisation with the additional channel can generate better results overall. Training a network in HSL colour space can be better than using RGB colour space and the RG method because it generates better images than the other techniques. Moreover, due to complexity reduction, HSL colour space spent less time training and sampling than RGB. As stated in Chapter 2, training a network in a colour space other than RGB could enhance the overall model performance with faster training and sampling with the benefit of having high-quality colour images. In conclusion, PixelCNN can effectively colourise greyscale images with the appropriate techniques and a suitable neural network architecture.

6.2 Limitation

Although PixelCNN is strong in capturing pixel dependencies, it is slow in sampling because it generates images pixel by pixel sequentially compared to other generative models that can generate parallel multiple pixels, especially when dealing with high-resolution images. For example, creating three samples from one RGB image takes 7 minutes because the network generates 64×192 pixels for each sample. However, for small-size samples, for instance, the datasets that are used in denoising noisy images take much less time to generate samples than RGB images, as the trained model generates 28×28. Therefore, the increase in pixels can increase the time complexity, making generating larger output time-consuming. Another limitation is that the predicted pixel value can affect the future pixel value, so an error in the pixel value can lead to less coherent and accurate samples.

Training PixelCNN models can be complex because capturing more dependencies requires an increase in the depth of the neural network and, thus, an increase in the training time. So, the choice of the depth of the neural network is vital to increase the accuracy of generating images. Another limitation of PixelCNN is that it has a blind spot that prevents it from capturing the dependencies of all the prior pixels. Because PixelCNN is a convolutional model, the convolutional layers only cover a receptive field of the applied kernel, so the target pixel would only have information from pixels in that receptive field [14]. Moreover, GPU memory availability could limit the training process because high-resolution images require more memory, which makes it necessary to resize these images to a smaller size. Memory availability also affects batch size, as high-resolution images take more memory, which means a smaller batch size can lead to slower training, as discussed in Chapter 3, section 3.1.3. The proposed architecture in Chapter 4 produces poor-quality images for the RG model without the additional channel, even though many experiments have been tried to enhance quality by increasing the network depth and number of epochs.

6.3 Future Study

Moving forward, PixelCNN can be expanded to denoise colour images and colourise different colour spaces, such as LAB, HSV, and YCbCr, which can depend on the chosen dataset. In order to produce better models, it is recommended to use variants of PixelCNN in these tasks, which can be conditional PixelCNN, Gated Pixel, and PixelCNN++. The autoregressive approach of PixelCNN allows it to solve other tasks, such as deblurring to restore blurry images and super-resolution to generate high-resolution images from low-resolution.

7. Conclusion

Image restoration is a crucial concept in computer vision as it plays a significant role in elevating the quality and clarity of digital images. It enables researchers to restore various degraded images by noise, blur, or any other degradation; in addition, it provides a way to restore other types of images, such as restoring colour information, greyscale images and old digital images. In resolving these complex challenges of image restoration, autoregressive models proved to be an effective solution for reconstructing high-quality images. The exploration into autoregressive models, especially PixelCNN, shows the effectiveness of autoregressive models in image restoration for degraded noisy images and colourising greyscale. The creation of an appropriate neural network and techniques in autoregressive models can create a strong model with superior image generation that has perfect accuracy for colourisation and denoising challenges.

Although PixelCNN can have limitations in the computing processes, it has the potential to restore images with high quality, which makes autoregressive models unleash significant potential for future studies. Future research should focus on all PixelCNN and its variants' capabilities in solving other complex problems, such as dealing with 3D images. In summary, PixelCNN's exploration of denoising and colourisation challenges opens many areas for application in computer vision; it can innovate several aspects of visual content.

References:

[1] O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G.V., Krpalkova, L., Riordan, D. and Walsh, J. "Deep Learning vs. Traditional Computer Vision" *Advances in Intelligent Systems and Computing*, vol. 943, pp. 128–144, Apr. 2019, doi: https://doi.org/10.1007/978-3-030-17795-9_10

[2] Z. Jin, M. Iqbal, Dmytro Bobkov, W.-B. Zou, X. Li, and E. Steinbach, "A Flexible Deep CNN Framework for Image Restoration," vol. 22, no. 4, pp. 1055–1068, Apr. 2020, doi: https://doi.org/10.1109/tmm.2019.2938340.

[3] J. Su, B. Xu, and H. Yin, "A survey of deep learning approaches to image restoration," *Neurocomputing*, vol. 487, pp. 46–65, May 2022, doi: https://doi.org/10.1016/j.neucom.2022.02.046

[4] K. Seetharaman, "A block-oriented restoration in gray-scale images using full range autoregressive model," *Pattern Recognition*, vol. 45, no. 4, pp. 1591–1601, Apr. 2012, doi: https://doi.org/10.1016/j.patcog.2011.10.020

[5] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, Jul. 2017, doi: https://doi.org/10.1109/tip.2017.2662206

[6] C. Saxena, D. Kourav, 'Noises and Image Denoising Techniques: A Brief Survey' *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, no. 3, pp.878-885 (2014)

[7] Z. Liu, W. Q. Yan, and M. L. Yang, "Image denoising based on a CNN model," *IEEE Xplore*, Apr. 01, 2018. https://ieeexplore.ieee.org/abstract/document/8384706/ (accessed Mar. 21, 2024)

[8] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," Aug. 2004, doi: https://doi.org/10.1145/1186562.1015780

[9] I. Zeger and S. Grgic, "An Overview of Grayscale Image Colorization Methods," Sep. 2020, doi: https://doi.org/10.1109/elmar49956.2020.9219019

[10] K. Nazeri, E. Ng, and M. Ebrahimi, "Image Colorization with Generative Adversarial Networks," *arXiv:1803.05400 [cs]*, vol. 10945, pp. 85–94, 2018, doi: https://doi.org/10.1007/978-3-319-94544-6_9

[11] A. Schaller, 'Generative Autoregressive Image Modelling using PixelCNN', *ResearchGate.* 2020. https://www.researchgate.net/publication/370222935_Generative_Autoregressive_Image_Modeling_using_PixelCNN (accessed Mar. 21, 2024)

[12] A. ven den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," *proceedings.mlr.press*, Jun. 11, 2016. https://proceedings.mlr.press/v48/oord16.html

[13] S. Guadarrama, R. Dahl, D. Bieber, M. Norouzi, J. Shlens, and K. Murphy, “PixColor: Pixel Recursive Colorization,” *arXiv (Cornell University)*, May 2017, doi: https://doi.org/10.48550/arxiv.1705.07208

[14] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves, “Conditional Image Generation with PixelCNN Decoders,” *Neural Information Processing Systems*, 2016. https://proceedings.neurips.cc/paper_files/paper/2016/hash/b1301141feffabac455e1f90a7de2054-Abstract.html (accessed Mar. 21, 2024).

[15] A. Kolesnikov and C. H. Lampert, “PixelCNN Models with Auxiliary Variables for Natural Image Modeling,” *proceedings.mlr.press*, Jul. 17, 2017. https://proceedings.mlr.press/v70/kolesnikov17a.html. (accessed Mar. 21, 2024).

[16] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a Convolutional Neural Network,” *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, Aug. 2017, doi: https://doi.org/10.1109/icengtechnol.2017.8308186

[17] Keiron O'Shea and R. R. Nash, “An Introduction to Convolutional Neural Networks,” *arXiv (Cornell University)*, Nov. 2015, doi: https://doi.org/10.48550/arxiv.1511.08458

[18] W. H. L. Pinaya, P. F. da Costa, and J. Dafflon, *Masking types in PixelCNN*. 2021. Accessed: Mar. 25, 2024. [Online]. Available: https://pedroferreiradacosta.github.io/post/auto_encoder/

[19] N. Ketkar, J. Moolayil, “Feed-Forward Neural Network”. In: *Deep Learning with Python,* 2nd ed. Apress, 2021, pp.93-132. doi: https://doi.org/10.1007/978-1-4842-5364-9

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, Massachusetts: The Mit Press, 2016

[21] D. Biswas *et al.*, “NOVEL GRAY SCALE CONVERSION TECHNIQUES BASED ON PIXEL DEPTH,” *Journal of Global Research in Computer Science Journal of Global Research in Computer Science Journal of Global Research in Computer Science Journal of Global Research in Computer Science*, vol. 2, no. 6, 2011, Accessed: Mar 26, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=32794dfc48b79e4fb387c9c4f5fae69318aa9e8c

[22] A. Ford, A. Roberts, and A. Roberts, “Colour Space Conversions.”, 1998, Accessed: Mar 26, 2024. [Online]. Available: https://poynton.ca/PDFs/coloureq.pdf

[23] R. Velastegui, L. Yang, and D. Han, “The Importance of Color Spaces for Image Classification Using Artificial Neural Networks: A Review,” *Lecture Notes in Computer Science*, Sep. 2021, doi: https://doi.org/10.1007/978-3-030-86960-1_6

[24] G. Saravanan, G. Yamuna, and S. Nandhini, “Real time implementation of RGB to HSV/HSI/HSL and its reverse color space models,” Apr. 2016, doi: https://doi.org/10.1109/iccsp.2016.7754179

[25] N. Ketkar, J. Moolayil, “Introduction to PyTorch”. In: *Deep Learning with Python,* 2nd ed. Apress, 2021, pp.27-92. doi: https://doi.org/10.1007/978-1-4842-5364-9

[26] X. Teng, Z. Li, Q. Liu, M. R. Pointer, Z. Huang, and H. Sun, “Subjective evaluation of colourized images with different colorization models,” *Color Research and Application*, vol. 46, no. 2, pp. 319–331, Nov. 2020, doi: https://doi.org/10.1002/col.22593

[27] D. Lee, S. Choi, and H.-J. Kim, “Performance evaluation of image denoising developed using convolutional denoising autoencoders in chest radiography,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 884, pp. 97–104, Mar. 2018, doi: https://doi.org/10.1016/j.nima.2017.12.050

[28] J. Nilsson and T. Akenine-Möller, “Understanding SSIM,” *arXiv.org*, Jun. 29, 2020, doi: https://doi.org/10.48550/arXiv.2006.13846

[29] Y. LeCun, C. Cortes, and C. Burges, "The MNIST Database of Handwritten Digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[30] Zalando Research, "Fashion-MNIST," GitHub repository. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

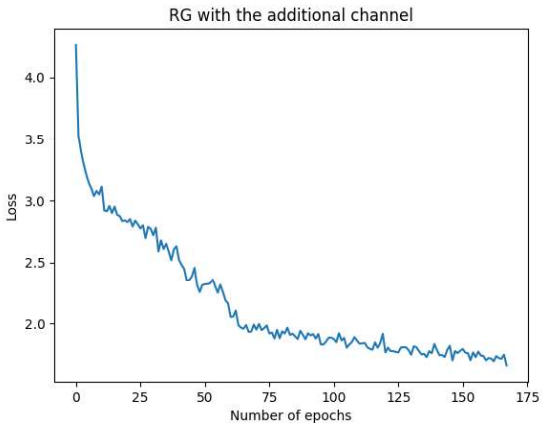
[31] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Master's thesis, Dept. of Computer. Sci., Univ. of Toronto, Toronto, ON, Canada, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

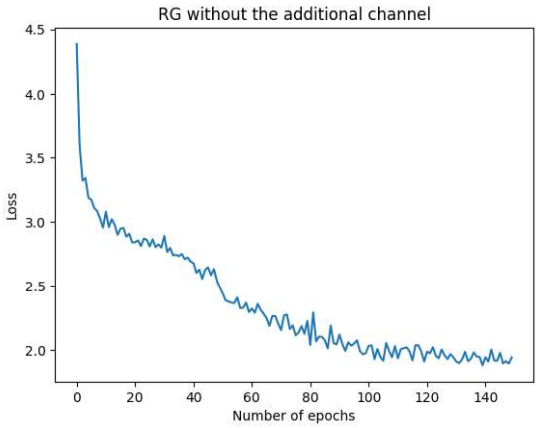
[32] University of Massachusetts, "Labeled Faces in the Wild (LFW)," [Online]. Available: http://vis-www.cs.umass.edu/lfw/

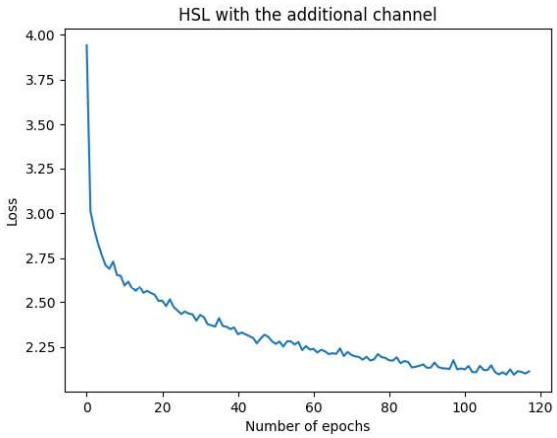
Appendices:

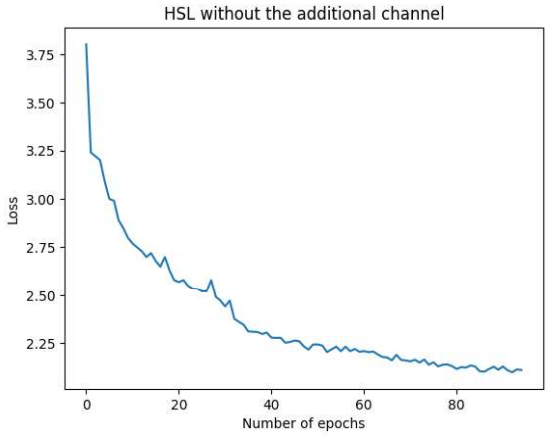
Loss Plots

The following graph represents the loss for each epoch during the training process for four models, that I managed to save.









HSL colour space conversion

The following two images represent the code of two functions that convert HSL to RGB and RGB to HSL.

1) RGB to HSL:

```
def rgb_to_hsl(img):

  min_value, _ = torch.min(img, dim=0,  keepdim=True)
  max_value, _ = torch.max(img, dim=0, keepdim=True)

  # Calculate Lightness
  l = (min_value  +  max_value)/ 2
  # calculate Saturation

  sat =  torch.where(l <= 0.5, (max_value-min_value)/(max_value + min_value), ( max_value-min_value)/(2.0-max_value-min_value))
  # if max == min
  sat =  torch.where(min_value == max_value, torch.zeros_like(max_value - min_value), sat)

  # Hue calculation
  hue = torch.zeros_like(max_value)
  hue = torch.where(max_value == img[0], (img[1]-img[2])/(max_value), hue)
  hue = torch.where(max_value == img[1], 2.0 + (img[2]-img[0])/(max_value), hue)
  hue = torch.where(max_value == img[2], 4.0 + (img[0]-img[1])/(max_value), hue)
  hue /= 6.0
  hue = hue % 1


  return torch.cat((hue,sat,l),dim=0)
```

2) HSL to RGB:

```
def hsl_to_rgb(img):

  hue, sat, l = img[0],img[1],img[2]

  temp_1 = torch.where(l<0.5, l*(1.0+sat), l + sat   l*sat)
  temp_2 = 2 * l - temp_1


  temp_r = hue + 1/3
  temp_g = hue
  temp_b = hue - 1/3

  def hue_convert(temp_1, temp_2, temp_c):

    temp_c = torch.where(temp_c < 0, temp_c + 1,
                         torch.where(temp_c > 1, temp_c - 1, temp_c))

    color = torch.where( temp_c < 1/6, temp_2 + (temp_1 - temp_2) * 6 * temp_c,
                        torch.where( temp_c < 1/2, temp_1,
                                    torch.where((temp_c < 2/3), temp_2 + (temp_1 - temp_2) * (2/3 - temp_c) * 6, temp_2)))

    return color

  r = hue_convert(temp_1,temp_2,temp_r)
  g = hue_convert(temp_1,temp_2,temp_g)
  b = hue_convert(temp_1,temp_2,temp_b)

  return torch.stack((r,g,b), dim=0)
```