ST311 Project
Colourising Greyscale Images with Autoencoders

Candidate Numbers: 36456, 37443

May 4, 2022

Abstract

Colourisation of black and white images has many useful applications, including providing richer context to historical photos and potentially accelerating the production of animation. We found literature which implemented convolutional autoencoders for image colourisation. [Baldassarre et al., 2017] proposed a model that combines deep convolutional neural network architecture with Inception-ResNet-v2 as a feature extractor. [Deshpande et al., 2016] used a variational autoencoder and constructed loss terms that 'avoid blurry outputs and take into account the uneven distribution of pixel colours'. In our study, we implemented our version of [Baldassarre et al., 2017]'s and [Deshpande et al., 2016]'s models, modifying the model with features such as L1 regularisation, Kaiming Initialisation, and batch training. We also propose our autoencoder architecture which is a simplistic model to act as a baseline for comparison. In summary, our results have found that Baldassare et al's model with slight modifications (model 1) performed the best overall. Nonetheless, we acknowledge our training was unfortunately handicapped by computing limits, most notably in the aspect of dataset size.

1 Introduction

Colourisation of black and white images has many useful applications, including providing richer context to historical photos and potentially accelerating the production of animation. However, in the case of historical photos, much research is required to accurately colour images by hand, and the process can be long and tedious. In such a colourisation process, typically one would identify the various objects present in an image by observing their shape, greyscale colour and relation to other objects, then colour the objects accordingly based on background and historical research. This motivates the use of neural networks to learn the patterns between shapes, objects and colours to help speed up and perhaps even improve the process of colourisation.

We found literature which implemented convolutional autoencoders for image colourisation. [Baldassarre et al., 2017] proposed a model that combines deep convolutional neural network architecture with Inception-ResNet-v2 as a feature extractor. [Deshpande et al., 2016] used a variational autoencoder and constructed loss terms that 'avoid blurry outputs and take into account the uneven distribution of pixel colours'. We were curious as to what factors make a good image colourisation convolutional autoencoder and how these two models would compare.

Thus, for our project, we implemented our version of [Baldassarre et al., 2017]'s and [Deshpande et al., 2016]'s models, then experimented with modifications to the architecture and training loop, such as the use of L1 regularisation, Kaiming Initialisation, and mini-batch training. We also propose our autoencoder architecture which is a simplistic model to act as a baseline for comparison.

In summary, our results have found that [Baldassarre et al., 2017]'s model with slight modifications (model 1) performed the best overall in terms of reconstruction loss and realisticness of output images, from training a subset of 2,048 images of our original training dataset of 24,996 nature-themed images. Nonetheless, we acknowledge that our training was unfortunately handicapped by Google Colaboratory hardware and usage allowances, most notably in the aspect of dataset size.

2 Methodology

This section explains our approach to the project and is supplemented by our code workbook, which was run on Google Colaboratory. References to relevant sections in the workbook are made in brackets.

2.1 CIELAB Colour Space

A general approach for running a neural network on images of dimensions H x W is to take the RGB values of each pixel's colour and present the RGB values over three channels, such that each image can be represented as a $3 \times H \times W$ tensor. However, the colourisation nature of our project meant we could reduce computation by using CIELAB ($L^*a^*b^*$) instead of RGB to represent each colour. Similar to RGB, the CIELAB colour space express colour on three axes: L^* is represented with values from 0 (black) to 100 (white), a^* indicates the red-green component of colour with $a^* > 0$ indicating red and $a^* < 0$ indicating green values, and b* indicates the blue-yellow components with $b^* > 0$ indicating yellow and $b^* < 0$ indicating blue values. Although theoretically the values a^* and b^* are unbounded, they are usually in the range $(-128, +127)$. The advantage of using CIELAB for our project is that instead of computing weights across all three red, green and blue channels, we only need to compute weights for the two channels a^* and b^*, since the values in L^* are known in a grayscale image, reducing computing costs significantly.

Although there is some evidence that CIELAB can outperform RGB in artificial intelligence tasks, it appears that which colour space performs better remains dataset and task-dependent. A study conducted by [Castro et al., 2019] on classifying Cape Gooseberries saw CIELAB outperforming RGB on support vector machine (SVM), artificial neural network (ANN), decision tree (DT), and k-nearest neighbours (KNN) models. However, [Reddy et al., 2017] tested colour spaces on CNN based AlexNet and found RGB performed better than CIELUV, a colour space similar to CIELAB. [Oza and Kumar, 2020] performed a similar test on a deep neural network and concluded that while convolutional neural networks (CNN) are not invariant to colour space, which colour space performs the best depends on other factors such as dataset and other specifications of the CNN model itself. Nevertheless, we believe that for our project, the computational gains from using CIELAB outweigh any potential performance loss.

2.2 Dataset

2.2.1 Dataset Choice

For our training data, we chose to use images from the Unsplash dataset, a dataset created by stock-photography sharing website `www.unsplash.com`. The dataset appears to be of overall high quality and is used by researchers at industry-leading institutions. It also has a good variety of images, with the full dataset containing images from over 250,000 contributing photographers and being the 'largest collaborative image dataset ever openly shared'. Furthermore, the dataset has useful accompanying features that ease usages, such as excellent documentation which includes data on colour coverage, and a convenient API that helps with image resizing. Last but not least, the licence for non-commercial use is free.

The Lite subset of the full Unsplash dataset was used due to computing limitations with Google Colaboratory, such that we cannot train too large of a dataset. With the lite dataset containing only nature-themed images, we narrowed down the scope of our training dataset with the hopes that our models could yield better performance by focusing on a particular image type. Conveniently, we believe that a model colouring nature-themed images should be easier to build than one for colouring man-made subjects, due to how there are relatively fewer possibilities for colourisation of nature-themed subjects (for example, the sky is usually blue, grey or black). As our project aims to compare and contrast various autoencoder models for colourisation, we believe the best comparisons can be found by maximising the chances of building generally good colourisation models.

2.2.2 Description of Dataset

The Lite Unsplash Dataset contains 24,996 nature-themed Unsplash images. Dimensions of the images in the dataset range from 563 to 16,384 pixels. Images have a range of aspect ratios from 0.34 to 4.13. The earliest submission date for images is 9 June 2013 and the latest submission date is 25 September 2020. Each image has a unique 11-character alphanumeric identifier.

2.2.3 Working Dataset Construction

Firstly, we downloaded (directly from the source) and resized all the images into a standardised square shape to fit into the encoder as inputs. Each image was resized into 256×256 pixels to balance computing constraints while still retaining enough detail. Resizing was done through the Unsplash API, which cropped each image to a square shape in the centre and subsequently performed downscaling. (Code Section 2.1 and 2.2) An assumption is made that the centre of each image holds the most colourful and interesting data which would make it ideal for training. Each image was then saved to a shared Google Drive folder.

Secondly, we filtered the dataset to obtain a dataset that we feel would be better for training. The large variety in the initial dataset meant we had images that were overall: 1. too dark, 2. too light or 3. too monotonous to be meaningful for training with the objective of colourising meaningful black and white images. If our dataset has too many of such images, it could lead to a high bias and low variance problem. We removed these images by making use of documentation on colour coverage available with the dataset, which details the top ten colours of every image in RGB values and the percentage coverage of each colour in the image. Based on the dataset, we constructed a weighted L^* metric below to measure the overall lightness/darkness of an image i:

$$L^{*(i)}_{\text{weighted}} = \sum_{j=1}^{10} L^*_j \times \%_{\text{coverage}_j}$$

Where j is a top ten colour of the image. We hypothesised that the problem of having images too dark is more severe than having images too light. Thus, we decided to remove more dark images than light images. Regarding images that were too monotonous, we removed images that had a single colour with at least 90% coverage. To summarise, we removed images that satisfy one or more of the following conditions:

1. Are one of the 1,000 images with the lowest L^* weighted values (too dark)
2. Are one of the 500 images with the highest L^* weighted values (too bright)
3. Images with single colour having over 90% coverage

This led to the removal of 1,705 images, leaving us with 23,291 images for our working dataset. (Code Section 2.4)

A CSV file of the Google Drive URLs to each of the 23,291 images was created to speed up access to images in the folder. (Code Section 2.5)

2.3 Data Loader

We created a custom PyTorch dataset class and data loader for our data. Our initial naive implementation can be found under Code Section 3.1. Due to its relatively slow running speed we created an improved version under Code Section 3.2, where multiprocessing was implemented to speed up the creation of the data loader. However, as the CPUs on Google Colaboratory are only dual-core, the increase in speed was unfortunately limited.

The dataset class and data loader preprocessed images according to the following steps:

1. Convert JPG image into a Numpy array of RGB values
2. Convert RGB values to CIELAB values
3. Normalise CIELAB values to be in range $(-1, 1)$
4. Seperate the L^* (perceptual brightness) channel from the a^*, b^* (colours) channels

The improved version of the dataset class and data loader utilised multiprocessing for the above image converter steps.

The data loader takes a given number of samples, train and validation split ratio and batch size as arguments to generate a trainloader and validloader for use in the training loop. Images are picked randomly from the working dataset to reduce the chances of overfitting.

2.4 Autoencoders

2.4.1 Models

We built, tested and compared following custom autoencoders on our dataset:

- Model 0: A simple convolutional autoencoder (Code Section 4.1.1)
- Model 0T: Model 0 but uses transposed convolution layers instead of convolution followed by upsampling layers (Code Section 4.1.2)
- Model 1: Modified version of model proposed by [Baldassarre et al., 2017], with the feature extractor omitted (Code Section 4.2.1)
- Model 2: Modified version of model proposed by [Deshpande et al., 2016]. The original model accepts a 64×64 tensor as input and predicts a $2 \times 64 \times 64$ tensor. Here we add additional convolutional layers to accept a 256×256 tensor as input and predict a $2 \times 256 \times 256$ tensor as output (Code Section 4.3.1)
- Model 2V: Variational autoencoder version of Model 2, with 64 dimensions in the latent space (Code Section 4.3.2)

2.4.2 General Structure

Each autoencoder consists of an encoder and a decoder, created with the sequential class from PyTorch's neural network package. All layers have a padding of 1, while layers either have strides of 1 or 2 depending on whether it is a downsampling layer in the encoder. The number of channels was increased in the encoder while decreased in the decoder, with the number being a power of two since this was observed often in similar models from literature.

2.4.3 Activation Function

Each convolution layer uses rectified linear unit (ReLU) activation, except for the final layer which uses the hyperbolic tangent activation function. ReLU activation was selected for its computational advantages and empirically promising results. Drawbacks such as the vanishing gradient problem in sigmoid or hyperbolic tangent (tanh) activation functions are not an issue with ReLU. The final layer uses tanh activation to scale outputs to range $(-1.0, 1.0)$, corresponding to the range of the two predicted channels a^* and b^*.

2.4.4 Kernel Size

A standard kernel size of 3×3 was selected for most of the models, in line with standard practices of selecting a square kernel with odd-numbered dimensions. Model 2 uses kernel size 5×5 for some of the encoder layers.

2.4.5 Transposed Convolution

We only used transposed convolution layers for model 0 as we realised that it performed much worse than using upsampling layers. We theorised this may be in part due to how transposed convolution could lead to 'uneven overlap' when the kernel size is not divisible by the stride. This shall be discussed further in the evaluation section.

2.4.6 Varational Autoencoder

A variational autoencoder version of model 2 was built as [Deshpande et al., 2016]'s paper built their model as a variational autoencoder. However, we did not trial VAE with other alternative models because we found its performance to be much slower and underwhelming compared to its non-variational counterpart.

2.5 Training Loop

This section details the reasoning behind choices made regarding various specifications of the model. (Code Sections 6.2 and 6.3)

2.5.1 Optimiser

The Adam optimiser was chosen for its efficiency with how it combines the strengths of both momentum and root mean squared propagation (RMSprop), making it well suited for parameter-heavy models such as ours.

2.5.2 Initalisation

The Kaiming initialization (also known as He initialisation) was chosen as it has been suggested that Kaiming initialization works better for ReLU activated layers, while Xavier for softmax activated layers. Similar to Xavier initialisation, Kaiming initialisation also applies the idea of balancing variance of weights for better training efficiency but does it specifically for the ReLU activation function. (Initially proposed by [He et al., 2015]) Although Xavier works well for softmax activated layers, [Kumar, 2017] suggests that it is ill-suited for ReLU.

2.5.3 Weight Decay

We chose to implement L1 regularisation over L2 regularisation as L2 moves coefficients towards zero but cannot be exactly zero, while L1 can. Adding the L1 penalty term discourages the model from overfitting the training data by penalising the sum of the absolute values of the weights. This imposes a sparsity constraint under L1 since coefficients can become exactly zero. Thus, we implemented weight decay through the use of a sparse loss function (Code Section 6.2), while setting weight decay (which in PyTorch is implemented as L2 regularisation) for our Adam optimiser to 0.

2.5.4 Batch Normalisation

Ultimately, none of the models included batch normalisation layers. Though theoretically batch normalisation can lead to faster convergence or enable the usage of a higher learning rate, trials on both model 0 and model 1 found no meaningful difference in convergence (Code Sections 4.1.3 and 4.2.2). We discussed this further in the evaluation section.

2.5.5 Learning Rate

A constant learning rate of $\lambda = 0.00001 = 1 \times 10^{-5}$ was chosen for the training of all models. Learning rates were trialled across a range of magnitudes to heuristically find an optimal value where there were relatively stable yet significant decreases in loss values across epochs. The relatively small value of λ meant the training speed for this model would be relatively slow.

2.5.6 Training and Validation Dataset Split

For each model, we used a sample of 2,048 images for training, of which 10% were reserved as the validation set. A relatively high proportion of data was used for training after taking other measures to prevent overfitting..

2.5.7 Mini Batch Size

A mini-batch size of 32 was used as a heuristically optimal tradeoff between computation speed and model learning quality. A study conducted by [Intel, 2017] on the optimisation of AlexNet using the CIFAR-10 dataset with the TensorFlow library suggests that batch size of the power of 2 anecdotally seems to enable processors to perform better. [Bengio, 2012] suggest B = 32 as a good default value.

2.5.8 Adaptive Regularisation

The trainer function was designed to be adaptive to different kinds of regularisation depending on the model being used:

1. Sparse Autoencoder, using L1 regularisation, with the formula below:

$$\text{Loss} = \text{MSE Loss} + \lambda \times \sum_i |w_i|$$

   where λ was chosen to be 0.001, and the regularisation term is the sum of the absolute value of all the weights in every layer of the encoder.

   Without any sparsity, an autoencoder neural network may be able to simply copy the inputs if trained for enough epochs, instead of extracting the important features from the inputs. L1 regularisation (compared to L2 regularisation) in a sense performs feature selection, as it is able to shrink the weights that are less important to the learning towards zero.

2. Variational Autoencoder, using Kullback-Leibler (KL) divergence, with the formula below:

$$\text{Loss} = \text{MSE Loss} + \alpha \, \text{KL}(\mathcal{N}(\mu, \sigma^2) \parallel \mathcal{N}(0, I))$$

   where a is a scaling factor due to exploding gradients encountered when training the model.

2.5.9 Early Stopping

To avoid overfitting the model by training for too many epochs, we implemented an early-stopping mechanism for the trainer. The trainer will terminate early if the validation loss keeps continuously increasing for a given number of epochs (defined as patience). We implemented patience = 10 for all of our models. (Code Section 6.2)

2.5.10 Progress and Performance Tracking

An epoch and mini-batch progress bar from the tqdm library was added to the training loop to track training progress. (Code Section 6.2) A modified version of an animator for visualising training and validation losses from D2L was also included (Code Sections 5 and 6.2). CSV writer was also used to log training and validation losses on a separate CSV file for performance tracking purposes.

2.5.11 Model Checkpoints

Checkpoints were utilised such that partially trained models can be saved and subsequently loaded to a new coding session to resume training. This was especially helpful given GPU constraints in Google Colaboratory and the lengthy training time required. (Code Section 6.2)

2.6 Displaying Results

A printer function was devised to convert tensor outputs from the autoencoder back to displayable images. This required combining the predicted channels from the autoencoder (a^* and b^*) back with L^*, and converting colours back to RGB such that it can be plotted with Matplotlib. (Code Sections 6.4, 6.5 and 6.6)

3 Numerical Evaluation using Real Data

This section displays results for our 5 models. We first present final reconstruction losses for all of our models, at a standardised epoch and standardised running time. Then, training and validation losses graphs were plotted to observe convergence across models. We then visualise the results of the models by running nature-themed images outside of the dataset through the models. To demonstrate the applicability of our model on greyscale images, we also ran greyscale historical images through the model and displayed the results. Finally, we discussed the results we have obtained and notable observations we found while running the models.

3.1 Summary of Final Reconstruction Losses

The table below compares mean square error loss for the different models at the 1000th epoch and after 1 hour of run time. Comparison across run time was made to better compare efficiency across models as different models are of different depth and thus have different runtime speed per epochs. We recorded the runtime required for each model to run for 100 epochs to subsequently calculate the epoch at which runtime would reach 1 hour, and displayed reconstruction loss for that epoch.

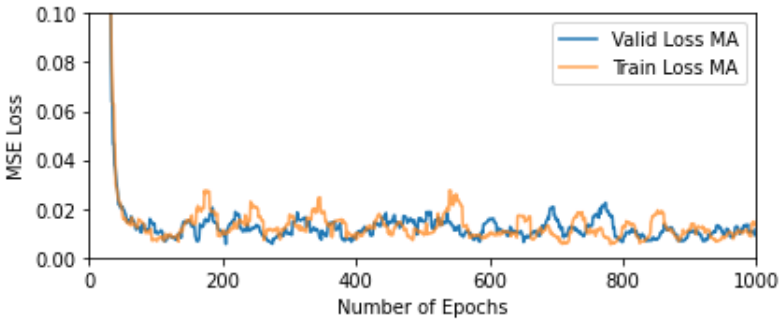
Due to splitting up the dataset into batches for training, we found that training and reconstruction losses had large fluctuations across epochs. Thus, for better comparison across models we present reconstruction losses as a moving average across the last 20 epochs.

Model	**Final Reconstruction Loss at Epoch 1000**	**Final Reconstruction Loss after 1 hour**
Model 0	$1.15653469 \times 10^{-2}$	$0.98272420 \times 10^{-2}$
Model 0T	$1.11266056 \times 10^{-2}$	$1.04134276 \times 10^{-2}$
Model 1	$0.70755723 \times 10^{-2}$	$2.93168066 \times 10^{-2}$
Model 2	$0.96528941 \times 10^{-2}$	$1.14109594 \times 10^{-2}$
Model 2V	$1.37207674 \times 10^{-2}$	$1.06512686 \times 10^{-2}$

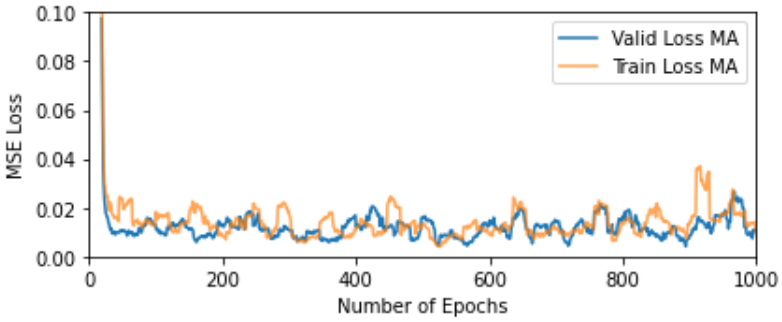
3.2 Graphs of Training and Validation Loss against Epoch

Graphs of training and validation loss against epoch are presented below for each model. Similarly, the losses are presented as a moving average across the last 20 epochs. For better comparisons the scale of the vertical axis has been fixed, thus high losses in the initial epochs were not captured in model 0.

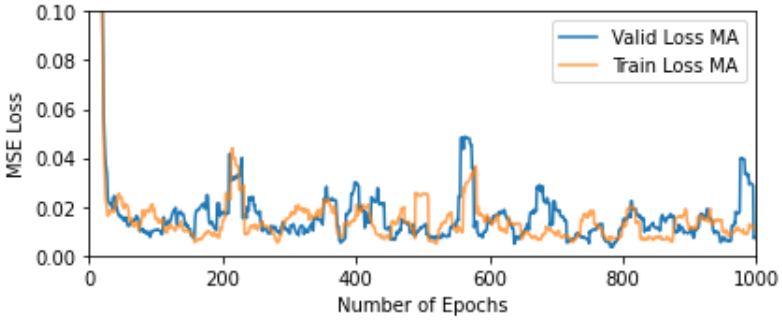
Model 0



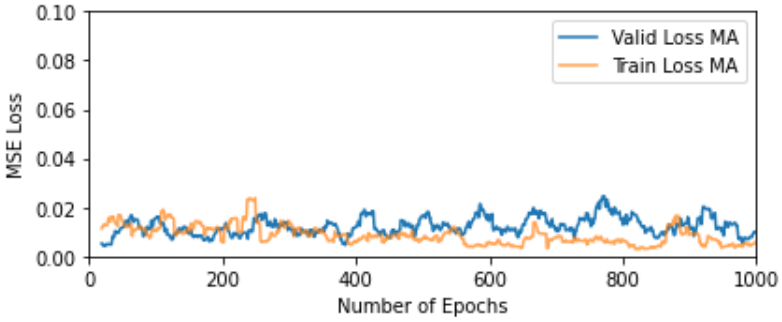
Model 0T



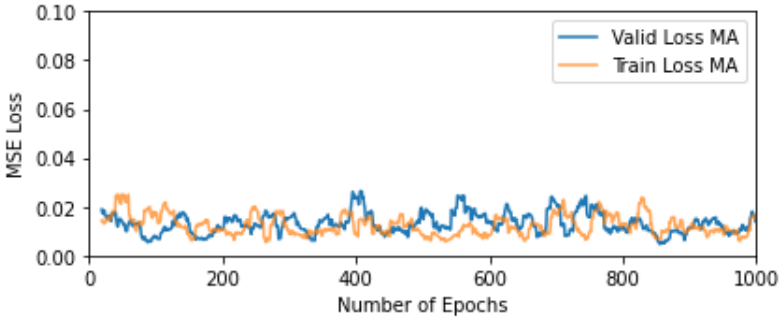
Model 1



Model 2



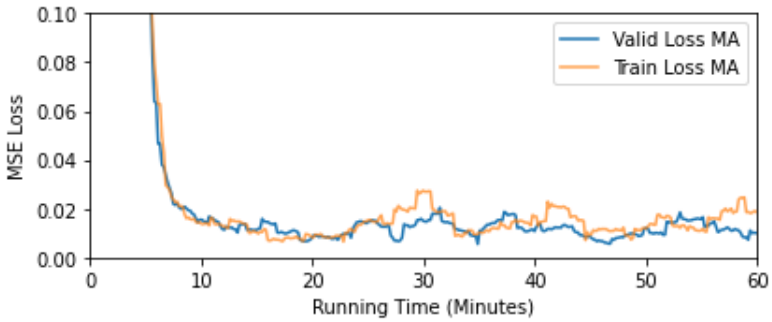
Model 2V



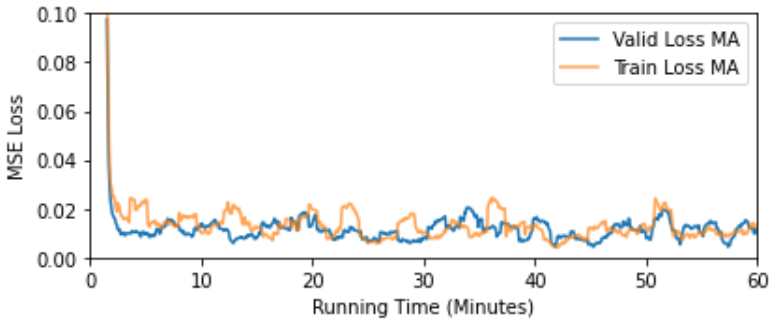
3.3 Graphs of Training and Validation Loss against Time

Graphs of training and validation loss against runtime are presented below for each model. Note that since the displayed losses are moving averages, there are no losses plotted for the first 20 epochs. Since running speed per epoch differs from model to model, these missing 20 epochs have varying impacts on the graphs for different models.

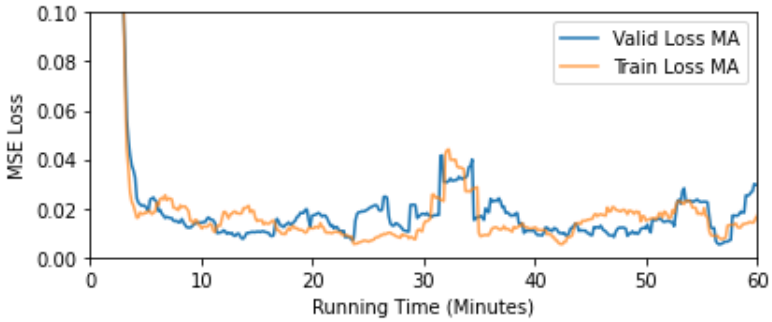
Model 0



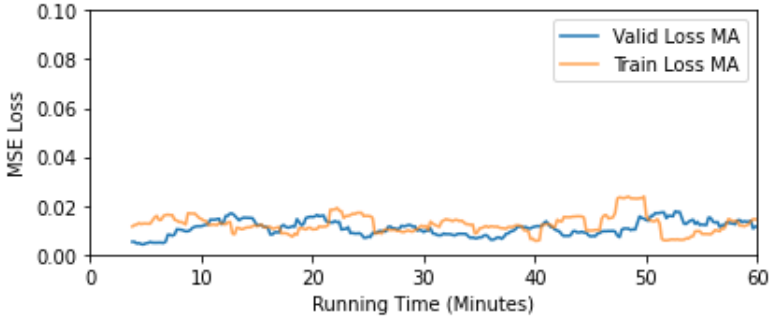
Model 0T



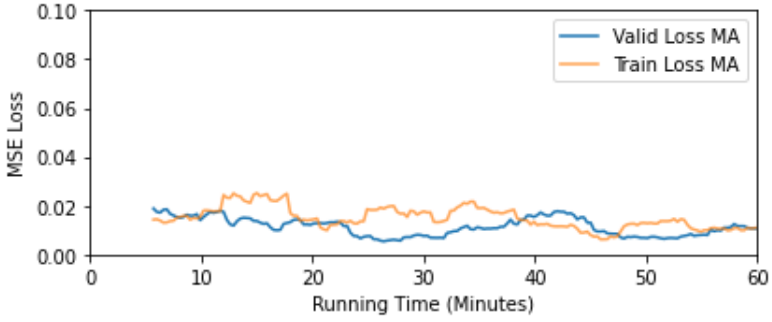
Model 1



Model 2

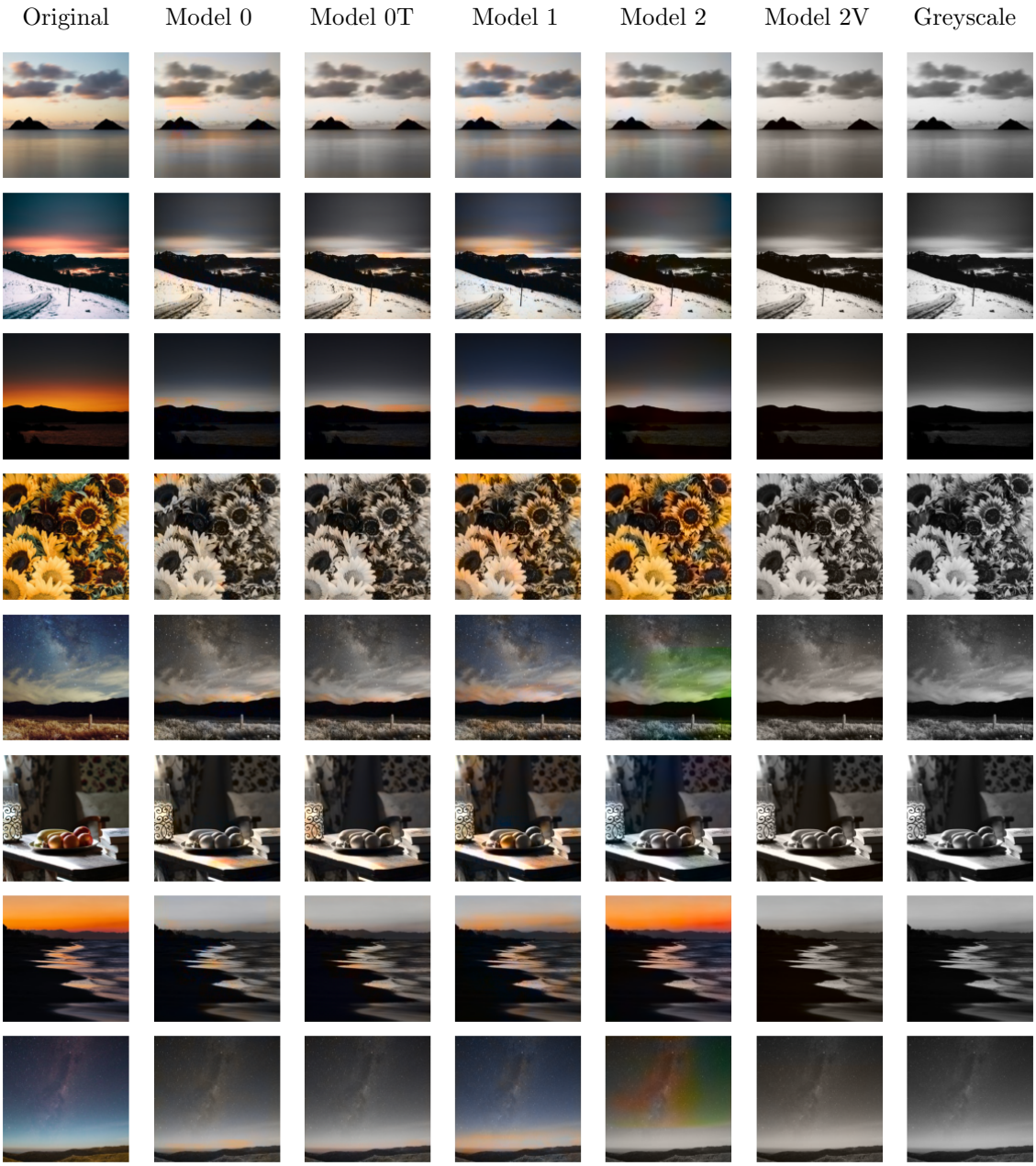


Model 2V



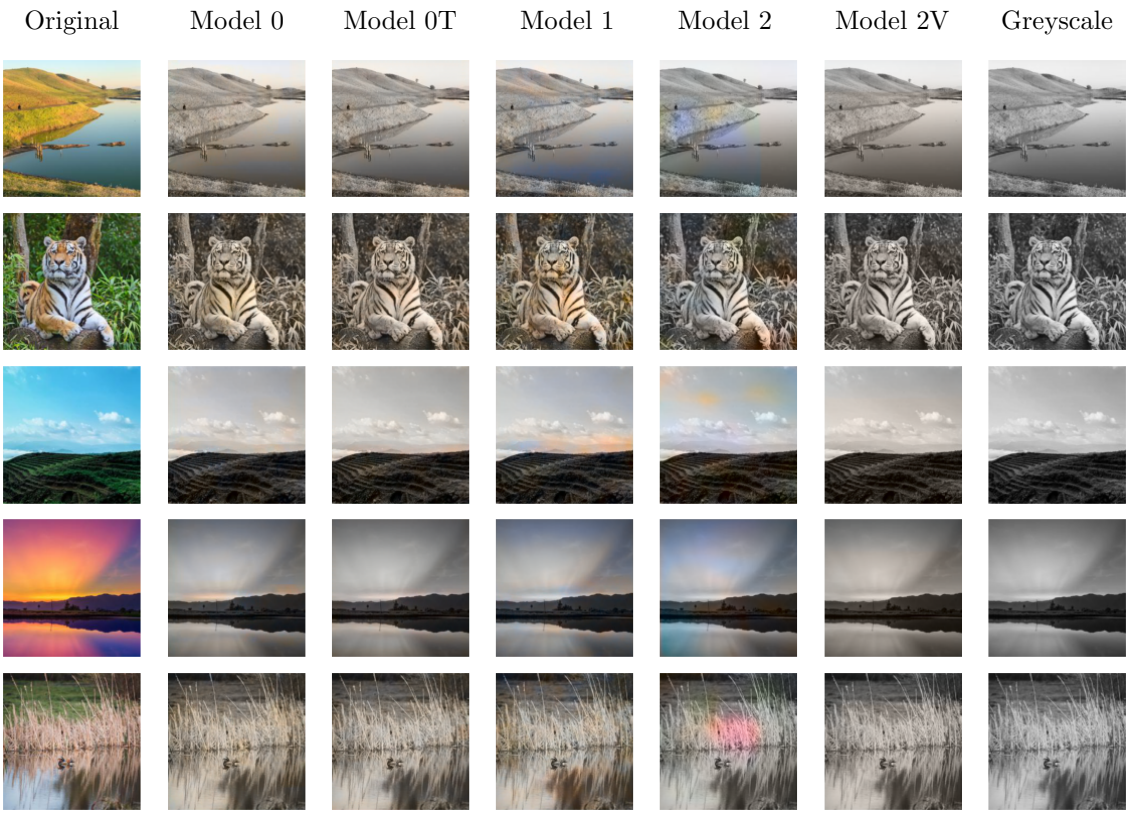
3.4 Results Visualised on Training Images

We visualised how well the models were able to learn our training data by passing training images through the trained models at the 1000th epoch and constructing output images with the output colour tensors.



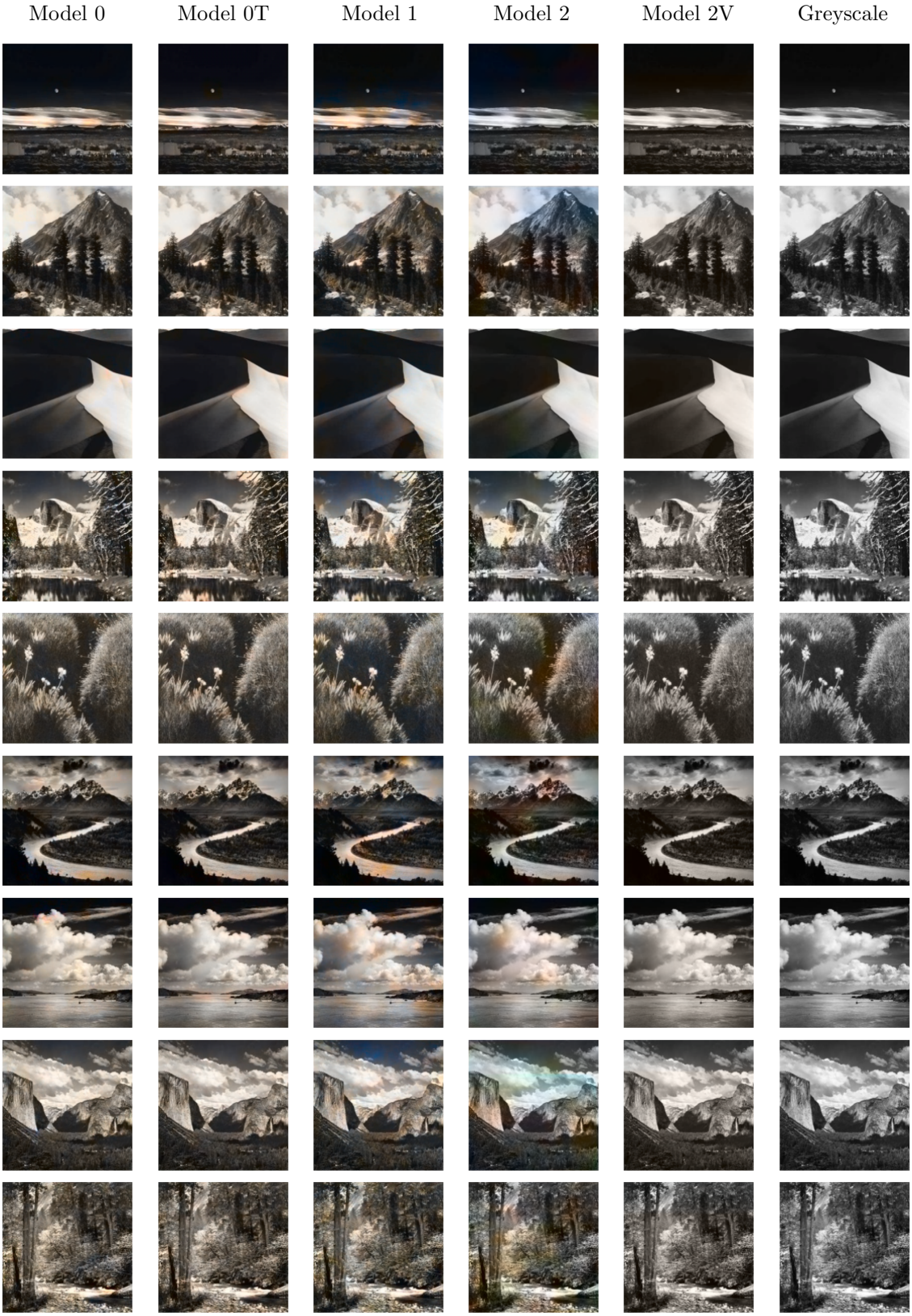
3.5 Results Visualised on Other Coloured Images

Similarly, visualisation of how well the models at the 1000th epoch can generalise to other coloured images is shown below. The images below were also obtained from unsplash.com, but only images published after 25 September 2020 were selected to avoid potential overlap with the training dataset.



3.6 Results Visualised on Other Greyscale Images

The following images are taken from [Probert, 2021]'s article on photographs by Ansel Adams, a famous American photographer.



3.7 Discussion of Results

3.7.1 Comparision Across Models

Final reconstruction losses shows that model 1 performed the best at the 1000th epoch, while model 0 performed the best after 1 hour of running time. This suggests that [Baldassarre et al., 2017]'s model performs well even when simplified onto our dataset. It also suggests that the shallow depth of model 0 could have potentially given it computational and efficiency advantages.

Model 1 appears to have a less stable convergence trajectory compared with the other models. Potentially, this could be due to a tradeoff between stability and accuracy in neural networks. In addition, model 2 seems to converge much quicker than models 0 and 1 in terms of epochs. This is likely due to the relative depth of model 2, which has 11 layers in the encoder, as opposed to models 0 and 1 with 5 and 8 layers respectively. Furthermore, we interestingly observed that model 0T converges faster than model 0, with 100 epochs of model 0 taking around 17 minutes to run while model 0T only takes 8 minutes. Further research and investigation may be required to understand precisely why.

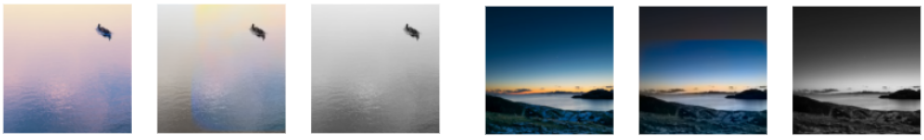
Results visualised on all images shows that model 1 and 2 appears to yield the most coloured outputs. Outputs for training images of models 1 and 2 appear a better reflection of reality than output for other images, suggesting potential overfitting. Nonetheless, for most of the images, besides some splotches of colour here and there, most of the colours on the images look like shades of brown. This is likely because brown is a relatively safe and middle colour (to create brown, a^* would be quite close to 0 while b^* would be a positive but small value) and thus choosing brown gives safe loss values. Though sometimes the splotches of colour appear somewhat correct (blue hues in the sky and sea for other coloured images: image 3, model 2, orange hues for the tiger's body for other coloured images: image 5, model 1), they can sometimes be very wrong (pink hues for other coloured images: image 2, model 2). Similarly, this suggests overfitting, and also an insufficiently large and rich dataset for the model to learn from. The model may have been able to pick up that the tiger's body should be orange if it has seen an image of a tiger in the training dataset, but if not it is unlikely to be able to do so.

3.7.2 The Variational Autoencoder

Unfortunately, the outputs from the variational autoencoder do not look very promising, even on the training dataset. There are effects from the model on images, but it simply looks like a brown filter has been placed across all the photos. The poor performance of the variational autoencoder for our dataset is further affirmed by model 2V having the highest reconstruction loss at the 1000th epoch. Potentially, variational autoencoders performed poorly for us due to our small dataset. From our understanding, the intended generative nature of a variational autoencoder means it attempts to learn how to colour images from a larger training space than an autoencoder by learning the probability distribution of how the data was generated. This means a much larger training space needs to be filled out, which perhaps means requiring much more data than the general autoencoder.

3.7.3 Discussion of Notable Observations During Training

It was observed that some of the training output would be partially coloured in one section but not coloured at all in the other, with a rather straight line dividing the two sections. This may have potentially arisen from how the models were trained on batches and updated with batch losses. The coloured parts of the image were perhaps more similar to the other images in the batch, but the non-coloured sections were more different. Thus, to minimise batch loss, the model colours only the more similar parts. Interestingly, the images below that display this phenomenon are also similar in how they are composed of more raw lines and colours, with identifiable objects taking up relatively less space. This might have made these images more at odds with other images within the same batch.



While trialling model 1 with batch normalisation, abnormal green coloured artefacts were observed in the training output. Unfortunately, we are uncertain regarding the reasons behind this phenomenon.

3.7.4 Evaluation of Methodology and Results

Though there are some differences in final reconstruction losses recorded, the differences were perhaps less dramatic than expected. In fact, all models tend towards a similar MSE loss of 0.02 after about 100 epochs and fluctuate around that level. It appears unlikely that this value will dip much lower given that loss has remained at about the same level for hundreds of epochs. This seems to suggest that our models are unsuccessful in training beyond the displayed level of accuracy. We examined the following reasons why:

Insufficient training data and insufficient epochs ran

Computational constraints of the Google Colaboratory GPU meant we could not load more than a few thousand images into the coding environment at once without the RAM being fully used up. The effect of insufficient training data can be seen in our visualised results. Furthermore, Google Colaboratory has overall usage limits and discourages inactive usage of its resources. This made it difficult, in our limited time available, to run our models for more epochs, even with model loading and saving functionalities. In comparison, [Baldassarre et al., 2017] trained their model on 60,000 images for 23 hours, while [Deshpande et al., 2016] trained on datasets varying from 13,233 to 126,277 images. Nonetheless, due to the convergence in loss observed from all models, the main problem likely does not lie with insufficient epochs. In fact, [Deshpande et al., 2016] only trained their models for 10 epochs.

Models are too shallow

Another potential weakness with our models is that they may be too shallow for sufficient training, as there are fewer weights for the model to learn. This is especially the case for our "naive" model 0. In many applications, it would appear that the deeper a model is, the better the model would perform. In general, a deeper network is less likely to overfit than a more shallow one. Unfortunately, Google Colaboratory computational constraints meant we were also handicapped in the model depth we could achieve. Nonetheless, the number of layers in our encoder and decoder for model 1 and model 2 follows the literature quite closely.

Autoencoder structure too simple

Aside from a variational autoencoder employed on model 2V, all our other models utilised a simple autoencoder structure with an encoder and decoder pair. Perhaps model performance could be improved by implementing further features in the autoencoder. For example, a feature extractor could be implemented in the bottleneck layer that could enable object recognition in images, similar to what [Baldassarre et al., 2017] have done. Similarly, image segmentation could be implemented to partition images into multiple sections for object identification. Furthermore, denoising can be applied to an autoencoder similar to how dropout can be applied on models that take one-dimensional inputs. [Bennouna et al., 2022] demonstrated that denoising autoencoders could perform better than variational autoencoders on small datasets.

Other Training Improvements

Batch Size: Using a batch size of 32 was perhaps too large. This caused issues such as fluctuating MSE loss is potentially the reason behind the partially coloured images. Using a smaller batch size may have yielded better results, though this comes with a computational speed tradeoff. [Masters and Luschi, 2018] suggested that best performances have been consistently achieved with minibatch sizes between 2 and 32.

Learning Rate: A fixed learning rate was used for all the models. Perhaps a variable learning rate may have helped with the stagnating learning rate in the models.

Batch Normallisation: The reason our initial batch normalisation trials did not work was because of the relatively small batch size used. It may also suggest that batch normalisation is more applicable to image classification problems as it can be used to highlight features in an object. Nonetheless, perhaps this could be investigated further and may help improve batch training.

Optimiser: Though Adam is a very popular optimisation algorithm, recent developments have found optimisers that could potentially outperform Adam, such as Adabound, suggested by [Luo et al., 2019]. Furthermore, [Wilson et al., 2017] suggests adaptive methods such as Adam generalise worse than stochastic gradient descent (SGD) methods. It may be worth exploring other optimisers to use for training.

4 Conclusion

4.1 Summary

In conclusion, our results have found that [Baldassarre et al., 2017]'s model with slight modifications (model 1) performed the best overall in terms of reconstruction loss and realisticness of output images, from training a subset of 2048 images of our original training dataset of 24,996 nature-themed images. Given that our version of [Deshpande et al., 2016]'s model (model 2) had more layers, this potentially suggests that a deeper network is not always necessarily better if the training dataset is small. Nonetheless, we acknowledge that our training was unfortunately handicapped by Google Colaboratory resource and usage allowances, most notably in the aspect of dataset size.

4.2 Directions for Future Research

Throughout the process of the study, we identified an array of different issues that could warrant further research. These were discussed in the discussion of the results section. A summary of them is as below:

- Trialling the various models we ran with a larger dataset. This was the main weakness of our exploration.
- Exploring deeper networks for colourisation models, both in the form of more channels and more convolution layers. [Goodfellow et al., 2016] suggested that deep autoencoders have many advantages, such as reducing computational cost, decreasing the amount of training data needed, and could yield better compression.
- Investigating computational differences between upsampling and transposed convolution, and identifying reasons as to why transposed convolution had a shorter running time for our model.
- Constructing neural networks with more complex autoencoder architecture, such as adding denoising and feature extractor.
- Examine other training specifications more carefully and run further trials across different specification combinations. These training specifications include batch size, learning rate, batch normalisation and training optimisers.

5 Statement about individual contributions

Contribution of 36456 = Contribution of 37443 = 50%

Technical contributions of Candidate No. 36456:

- Code Sections: Section 1, Section 2, Section 4.1, Section 5, Section 6.2, Section 6.3
- Report: Abstract, Introduction, Methodology, Discussion of Results, Conclusion

Technical contributions of Candidate No. 37443:

- Code Sections: Section 3, Section 4.2, Section 4.3, Section 6, Section 7
- Report: Methodology, Numerical evaluation using real data, Discussion of Results

References

[Hru, 2017] (2017). *When and Why Are Deep Networks Better Than Shallow Ones?*

[Baldassarre et al., 2017] Baldassarre, F., Morín, D. G., and Rodés-Guirao, L. (2017). Deep koalarization: Image colorization using cnns and inception-resnet-v2.

[Bartoldson et al., 2019] Bartoldson, B. R., Morcos, A. S., Barbu, A., and Erlebacher, G. (2019). The generalization-stability tradeoff in neural network pruning.

[Bengio, 2012] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures.

[Bennouna et al., 2022] Bennouna, K., Chougrad, H., Khamlichi, Y., El Boushaki, A., and El Haj Ben Ali, S. (2022). Variational autoencoders versus denoising autoencoders for recommendations. In Bennani, S., Lakhrissi, Y., Khaissidi, G., Mansouri, A., and Khamlichi, Y., editors, *WITS 2020*, pages 179–188, Singapore. Springer Singapore.

[Brownlee, 2019] Brownlee, J. (2019). A gentle introduction to activation regularization in deep learning.

[Brownlee, 2020] Brownlee, J. (2020). How to control the stability of training neural networks with the batch size.

[Castro et al., 2019] Castro, W., Oblitas, J., De-La-Torre, M., Cotrina, C., Bazán, K., and Avila-George, H. (2019). Classification of cape gooseberry fruit according to its level of ripeness using machine learning techniques and different color spaces. *IEEE Access*, 7:27389–27400.

[Deshpande et al., 2016] Deshpande, A., Lu, J., Yeh, M.-C., Chong, M. J., and Forsyth, D. (2016). Learning diverse image colorization.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Graphic Quality Consultancy, 2021] Graphic Quality Consultancy (2021). Introduction to colour models ("spaces").

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[Intel, 2017] Intel (2017). Cifar-10 classification using intel® optimization for tensorflow*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

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

[Kumar, 2017] Kumar, S. K. (2017). On weight initialization in deep neural networks.

[Luo et al., 2019] Luo, L., Xiong, Y., Liu, Y., and Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate.

[Masters and Luschi, 2018] Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks.

[Maxim, 2017] Maxim (2017). What's the difference between variance scaling initializer and xavier initializer?

[Minaee et al., 2020] Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2020). Image segmentation using deep learning: A survey.

[Ng, 2011] Ng, A. (2011). Lecture notes on sparse autoencoders.

[Odena et al., 2016] Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

[Osendorfer et al., 2014] Osendorfer, C., Soyer, H., and van der Smagt, P. (2014). Image super-resolution with fast approximate convolutional sparse coding. In Loo, C. K., Yap, K. S., Wong, K. W., Beng Jin, A. T., and Huang, K., editors, *Neural Information Processing*, pages 250–257, Cham. Springer International Publishing.

[Oza and Kumar, 2020] Oza, U. and Kumar, P. (2020). Empirical examination of color spaces in deep convolution networks. *International Journal of Recent Technology and Engineering (IJRTE)*, 9(2):1011–1018.

[Probert, 2021] Probert, G. (2021). 25 famous photographs by ansel adams (amp; 6 fun facts).

[PyTorch, 2022a] PyTorch (2022a). Datasets & dataloaders.

[PyTorch, 2022b] PyTorch (2022b). Saving and loading a general checkpoint in pytorch.

[Rath, 2020] Rath, S. R. (2020). Sparse autoencoders using l1 regularization with pytorch.

[Reddy et al., 2017] Reddy, K. S., Singh, U., and Uttam, P. K. (2017). Effect of image colourspace on performance of convolution neural networks. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 2001–2005.

[Versloot, 2022] Versloot, C. (2022). He/xavier initialization & activation functions: choose wisely.

[Wilson et al., 2017] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning.

[Zhang et al., 2021] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.

[Zhang et al., 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization.

[Zhou, 2018] Zhou, S. (2018). What happens in sparse autencoder.