– COS429 Written Report, Spring 2023 –

# Image Colorization

Gedeon Guercin and Erkhem Unenbat
Project Mentor: Maxine Perroni-Scharf

Project Codebase: https://github.com/GedeonGuercin/COS429-Final-Project

## Abstract

*Image colorization is the process of adding color to grayscale or black and white images. This technique has been used in various applications, including historical photo restoration, movie colorization, and video game design. The goal of image colorization is to produce natural and aesthetically pleasing color images that are as close as possible to the original scene.*

*There are several approaches to image colorization, including manual colorization, rule-based methods, and deep learning-based methods. Manual colorization involves manually selecting and applying colors to different parts of the image. Rule-based methods use predefined rules and heuristics to infer color information based on the image context. Deep learning-based methods, on the other hand, use neural networks to learn the color mapping function from a large dataset of color images.*

*In recent years, deep learning-based methods have shown promising results in image colorization, achieving state-of-the-art performance in terms of color accuracy and image quality. These methods typically use convolutional neural networks (CNNs) or generative adversarial networks (GANs) to learn the color mapping function. However, image colorization remains a challenging task, especially for complex scenes with multiple objects and textures.*

*In summary, image colorization is a useful technique that can enhance the visual appeal and historical significance of grayscale images. Deep learning-based methods have shown great potential in this field, but further research is needed to improve the robustness and efficiency of these methods for real-world applications.*

## 1. Introduction

Given a set of grayscale images, we wanted to attempt to colorize the images to the best approximation. By doing this, we could take old photos in grayscale and visualize them in a contemporary context with colors. The main idea of this project is: how can we utilize CNNs to generate color images?

## 2. Related Work

Over the years, there have been numerous studies and efforts to tackle the problem of image colorization. One of the early approaches used clustering techniques to segment the image into regions and assign colors based on the most frequently occurring color in each region. This method, while simple, produced colorized images that lacked fine detail and realism.

In 2016, Iizuka et al. offered End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification [1]. Their method showed promising results on a range of images, including portraits and landscapes.

More recently, deep learning approaches have emerged as the dominant method for image colorization. CNNs have been used to learn the mapping between grayscale images and their corresponding color images. One of the earliest CNN-based methods for image colorization was developed by Zhang et al. in 2016, where they introduced a fully convolutional network architecture called ColorNet [4]. Called Colorful Image Colorization, they approached the problem as a classification task and they also considered the uncertainty of this problem (e.x. a car in the image can take on many different and valid colors and we cannot be sure about any color for it). They proposed a fully automatic approach that produces vibrant and realistic colorizations. They tackled it as a classification task and used class-rebalancing at training time to increase the diversity of colors in the result. The system is implemented as a feed-forward pass in a CNN at test time and is trained on over a million color images from ImageNet.

We also looked at another project by Luke Melas-Kyriazi that approached the problem as a regression task [3]. This project implements a deep convolutional neural network for automatic

colorization, the problem of converting grayscale input images into colored images. The model is based on the ResNet-18 classifier and trained on the MIT Places365 database of landscapes and scenes. We borrowed some helped functions from Kyriazi in order to display our output images. Both the regression and classification used the LAB. This colorspace contains exactly the same information as RGB, but makes it easier to separate out the lightness channel from the other two (A and B).

More recently, some researchers have explored the use of adversarial training to improve the quality of colorization. For example, Isola et al. introduced a method called Pix2Pix, which uses a conditional GAN to generate colorized images from grayscale inputs [2]. Their method produces impressive results, particularly on images with complex textures and structures.

Overall, the recent advancements in deep learning have significantly improved the quality of automatic image colorization. However, there is still room for improvement, particularly in terms of preserving fine details and producing realistic colorizations of challenging images.

## 3. Implementation and Evaluation

This section includes the implementation of models as well as the quantitative and qualitative results. We began by using the MIRFLICKR25k from kaggle. It contained 25,000 images of size 224x224 in the the *LAB* color space. Images were separated by color space, each picture consisting of two files, one for the *AB* and another for the *L* color space. *L* is the grayscale (or lightness metric) version of the MIRFLICKR25k dataset, and it is meant to be used as the input to our model. The *A* channel contains information in the green to red color spectrum while the *B* channel contains information in the blue to yellow color spectrum. When the *L* and *AB* image channels are combined, a full color image is produced. Some base images of the MIRFLICKR25k data set can be seen in Figure 1.

We decided to implement a convolutional neural network, or a CNN. Our base model was originally pretty shallow, consisting of only 5 layers, and only went up to 128 channels. As with previous colorization models, our model also had to output images of the same height and

**Figure 1: Subset of MIRFLICKR-25000**

width as the input, the only difference being the number of color channels. At first, the image is downsampled as the model gets deeper. Then the model needs to upsample the image until it contains enough information to be outputted. At first, we dismissed this quality, but as we tested and troubleshooted the model, we realized that this was the only viable approach.

This model had minimal hyperparameter tuning as we wanted to observe how the base model would fare against the MIRFLICKR-25000 images. The hyperparameters can be seen in Table 1.

| Learning Rate | Epochs | Optimizer |
|---|---|---|
| 0.0001 | 25 | SGD |

**Table 1: Intial Model Hyperparameters**

The model architecture can be seen below in Figure 2.

The base model did not perform very well. This model was trained on 20,000 images with a validation size of 5,000. for 25 epochs. We use a batch size of 32 for the training and 10 for the validation. We continued to use these batch sizes as we noticed that training on larger batch sizes would result in memory usage errors. The output can be seen in Figure 3.

As seen in Figure 3, the base model produces minimal colorization. Although there is an obvious difference in the input image and the prediction, our model was simply not complex enough to learn and accurately output colors. It seemed to predominantly color the images in the yellow color space and also randomly add red and blue blotches. Although this looks better than the input

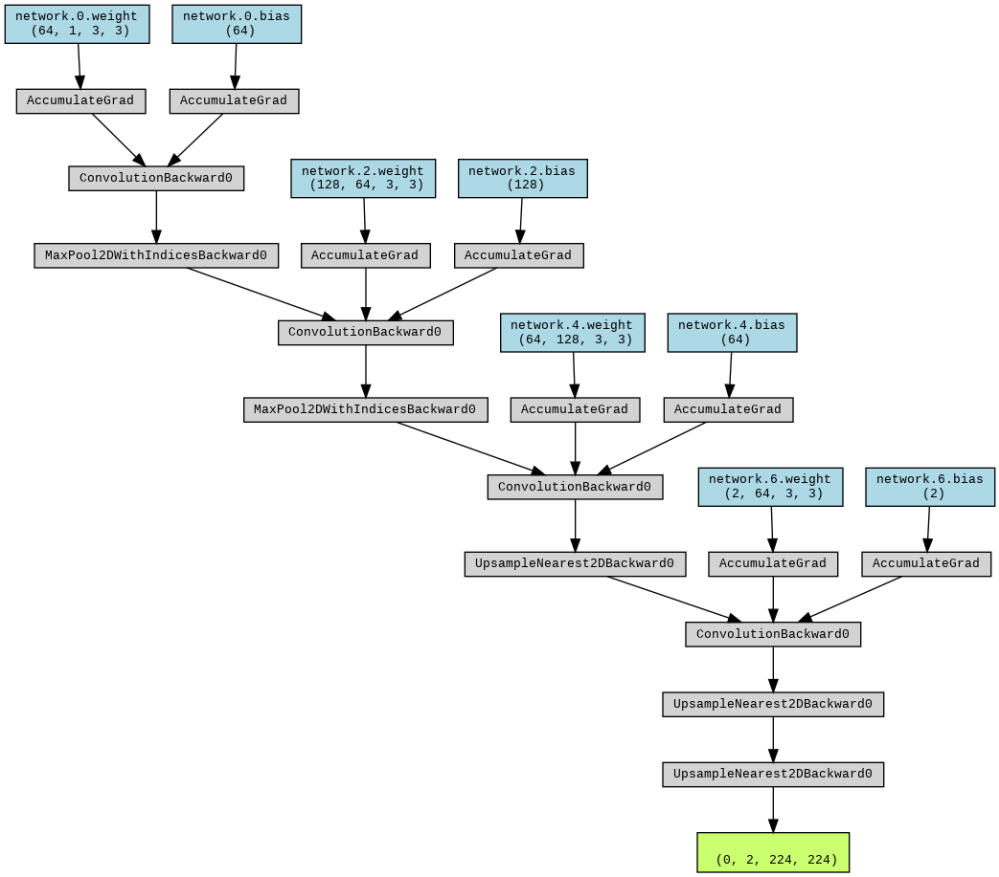


**Figure 2: Base Model Architecture**

images, this is not the result we were looking for.

The loss plots for the training and validation can be seen in Figure 4 and Figure 5 respectively.

As can be seen in Figure 4, the loss seems to stop decreasing on the training set after about 5 epochs. Similarly, the validation loss in Figure 5 stops decreasing very early, at around 10 epochs. Here we are using SGD, or stochastic gradient descent for our optimizer, with a momentum of 0.9. We took note of this poor performance and decided to use the Adam optimizer from here on. See section 3.1 for more details.

From this, we began to increase the complexity of our model by adding more layers, increasing the channels to 512, and adding batch normalization. Our final model architecture can be seen in Figure 6.

After we updated the model architecture, we trained it for 25 epochs on the full data set (20,000 for training and 5,000 for validation) with a batch size of 32 for the training and 10 for the

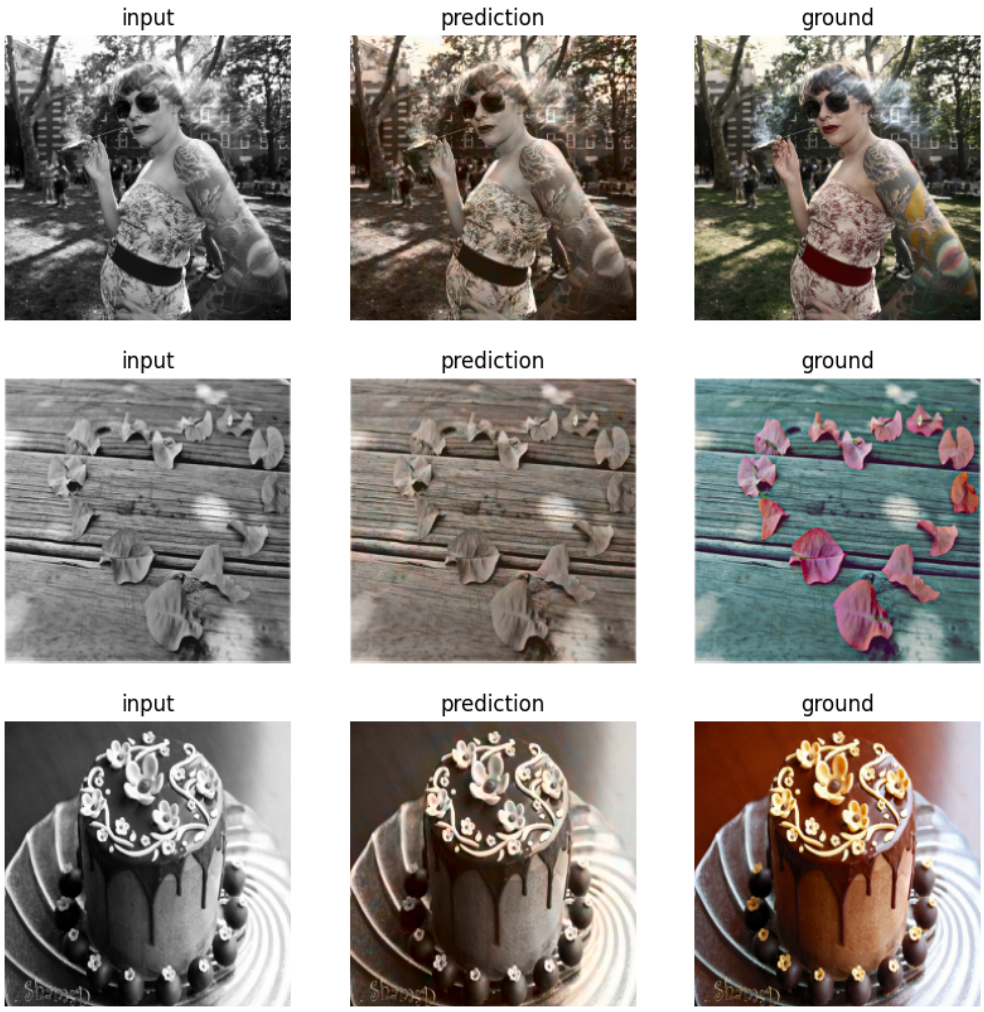


**Figure 3: Base Model Output**

validation. The output can be seen in Figure 7.

The loss plots for the training and validation can be seen in Figure 8 and Figure 9 respectively.

As seen in Figure 7, our updated model significantly improved the image colorization on the base images. We noticed that the prediction images began to show more distinct colors. When compared to the first model, we see prediction images not only in the blue-yellow color space, but also we begin to see images in the green-red color space. However, when it came to images like the one with the heart-shaped flowers, it becomes apparent that the model cannot distinguish and add distinct colors for each specific object. Instead, we see an almost random combination of red, green,

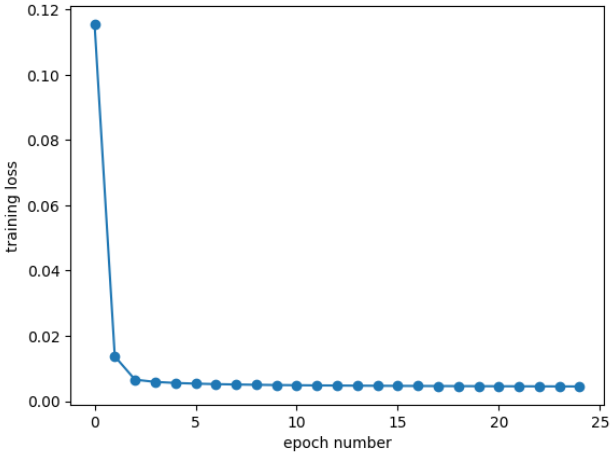


**Figure 4: Training Loss Plot**

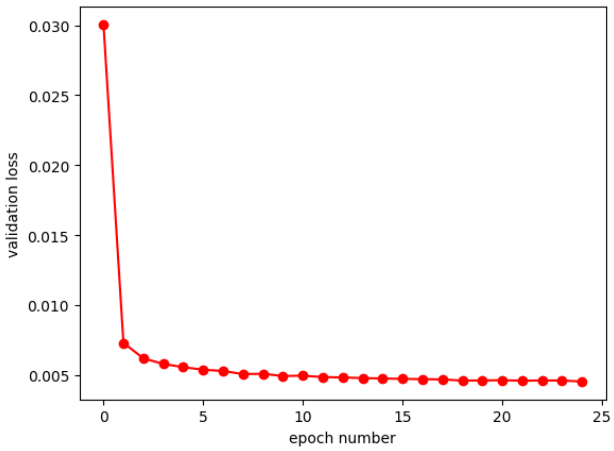


**Figure 5: Validation Loss Plot**

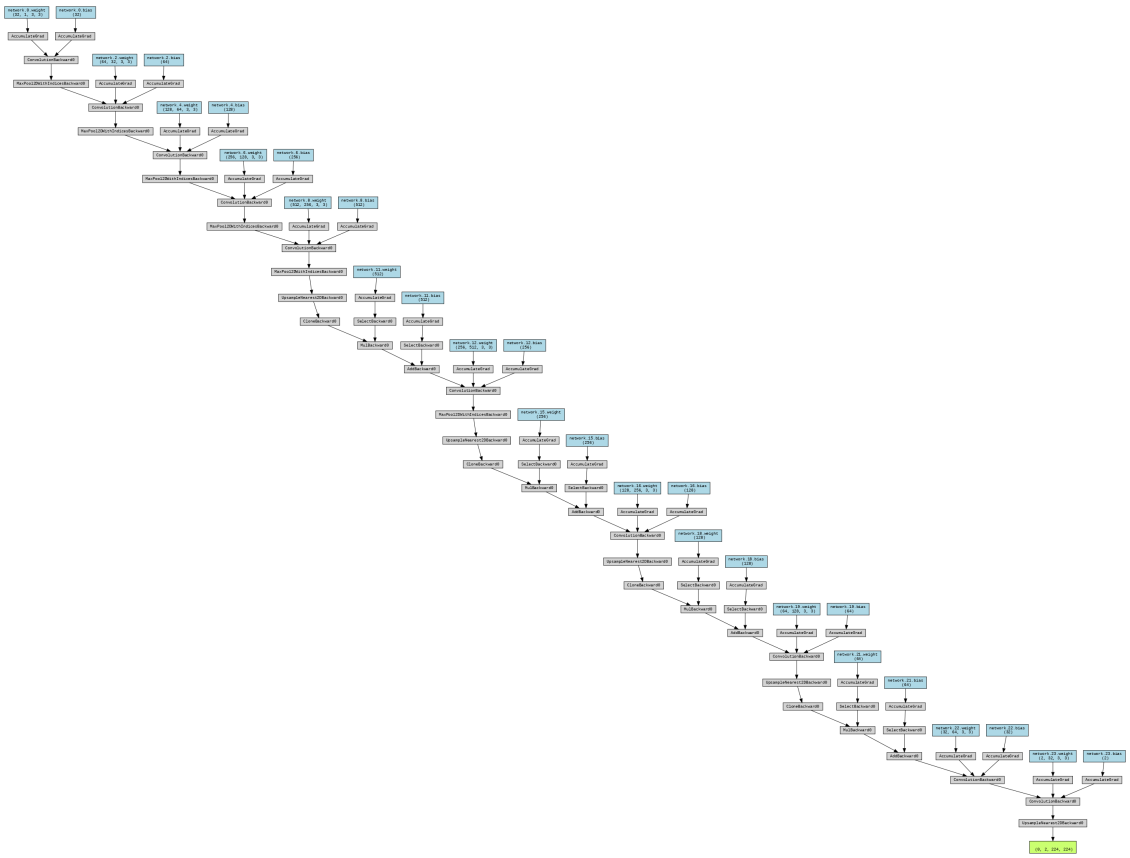

**Figure 6: Final Model Architecture**

and blue. For the first image, the model does a good job at distinctly coloring the area of the grass, even through the woman. In image three, the model seems to distinguish the cake from the plate. Overall, the model would predict colors in the correction color space, but had trouble coloring and distinguishing different objects for more complex image compositions. As can be seen in Figure 8,

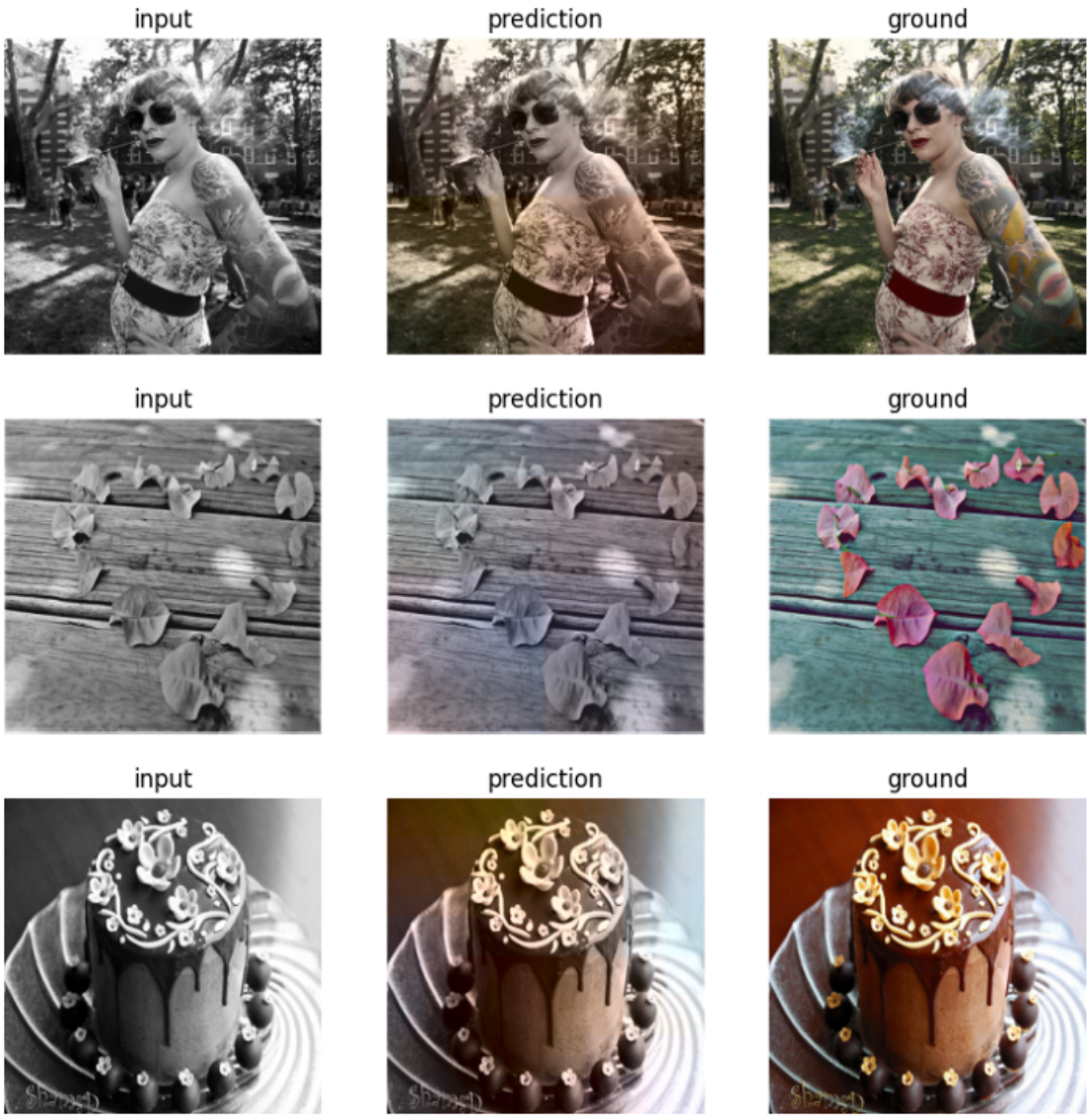


**Figure 7: Final Model Output on Base Images**

the training loss decreases at a much more gradual rate than that of the intial model. The validation loss in Figure 9 generally decreases as we train with more epochs. This indicates that the model is actually learning when compared to our initial model. This model, along with the new Adam optimizer proved to give us much more promising results. More

We then began to train our model with a greater number of epochs. In this case, we trained for 100 epochs. The results are shown below in Figure 10.

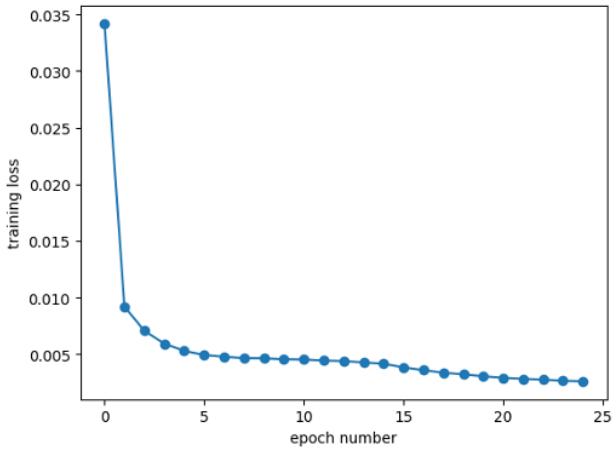


**Figure 8: Training Loss Plot**

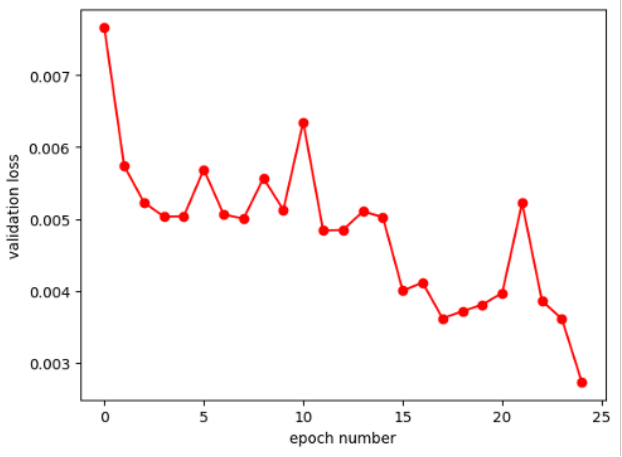


**Figure 9: Validation Loss Plot**

The loss plots for the training and validation can be seen in Figure 11 and Figure 12 respectively.

The hyperparameters can be seen in Table 2. From the output image in Figure 10,

| Learning Rate | Epochs | Optimizer |
|---|---|---|
| 0.0001 | 100 | Adam |

**Table 2: Final Model Hyperparameters**

we can see that the model works astonishingly well. The colors are much more accurate and they are generally colored in the correct spots. Prediction image three for the cake shows a striking resemblance to the ground image. The color is very close to being the same, and the cake is colored separately from the plate. For image 1, the grass and trees are colored green around the lady, and the lady is separately colored brownish red. For image two, while the results were a little disappointing, we see that the blueish-green color of the wood is almost replicated in the prediction image. As the composition is a little complex when compared to the other images, the model fails to color in the pink flower petals. The training loss plot in Figure 11, looks similar to the previous loss plot at Figure 8., except that the loss stops decreasing at around 100 epochs. The validation loss in Figure 12 generally decreases as we train with more epochs, and it is revealed that it also smooths out at around 100 epochs. More output images from our model on the MIRFLICKR-25000 dataset can be seen in Figure 13.

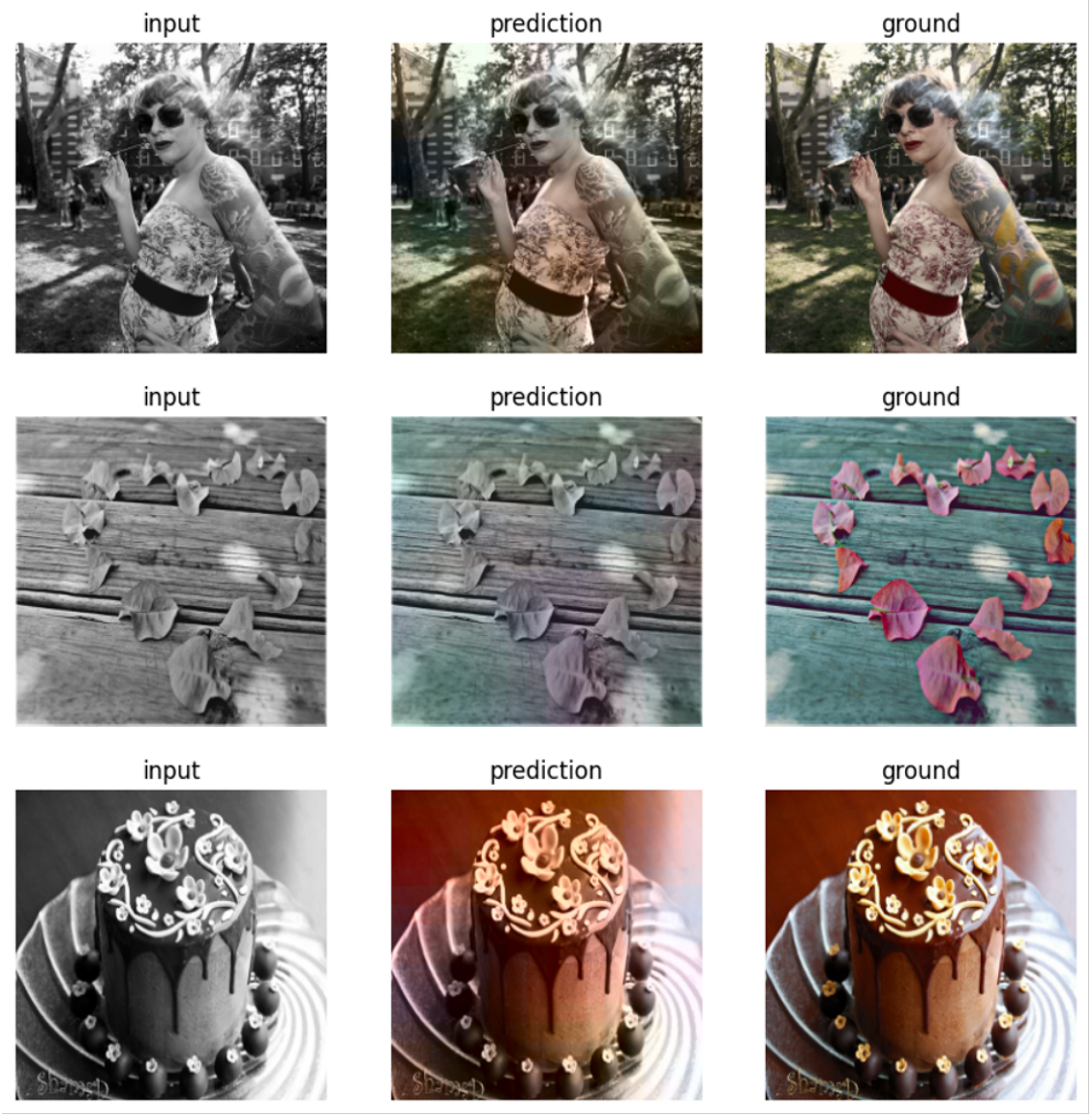


**Figure 10: Final Model Output on Base Images**

Following this, we considered implementing our model on different data sets in order to see if there was any noticeable difference. We decided train our model on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 RGB colour images in 10 classes, with 6000 images per class. We first had to convert these images to the LAB color space, and then separate the channels to produce both an L and an AB channel image for each image. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each

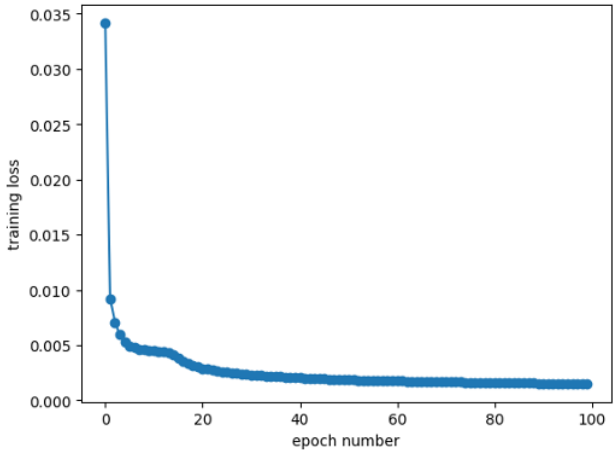


**Figure 11: Training Loss Plot**

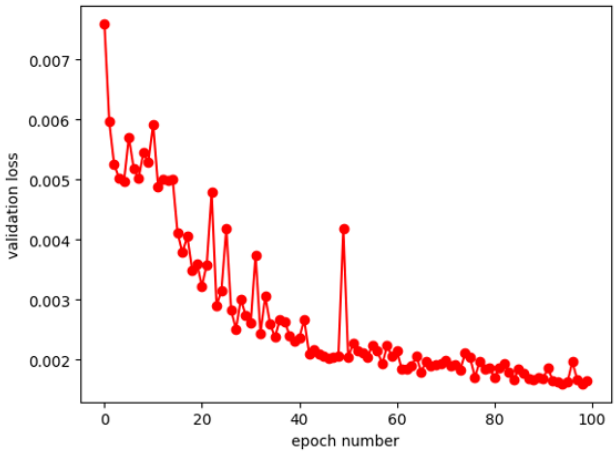


**Figure 12: Validation Loss Plot**

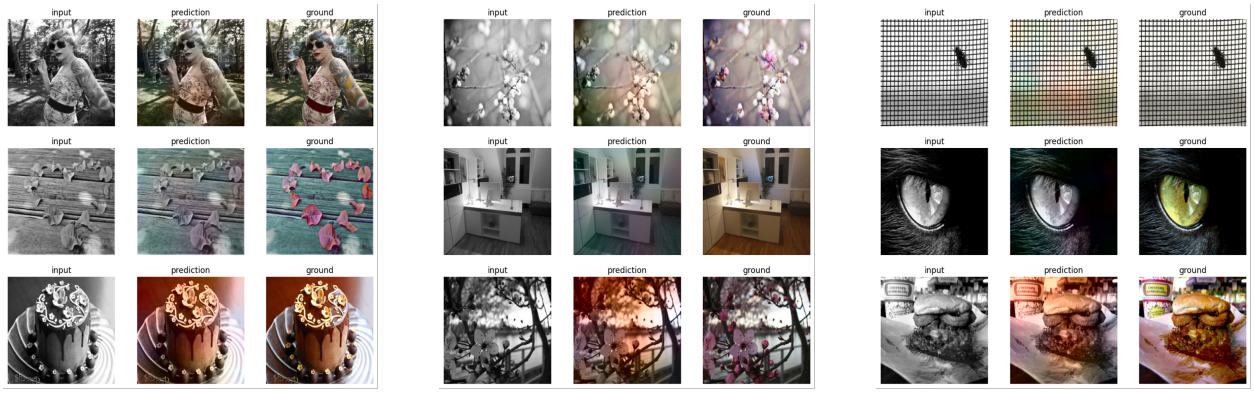


**Figure 13: Subset of MIRFLICKR-25000 on Final Model**

class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. These images were a lot smaller than the MIRFLICKR-25000 images, which were 224 by 224. By using smaller images, it allowed us to see the individual pixels in order to see how precise our model was coloring. When compared to the MIRFLICKR dataset, CIFAR-10 is less random since it has 6000 images in 10 different classifications. The images in the MIRFLICKR datset have little to do with each other. The intuition here is that the model can train on many different objects that may be colored the same way.

We trained our model on CIFAR-10 and used the same hyperparameters as shown in Table 2. The output of the model on some CIFAR-10 base images can be seen in Figure 14 and Figure 15.

The loss plots for the training and validation of the CIFAR-10 data set can be seen in Figure 16 and Figure 17 respectively.

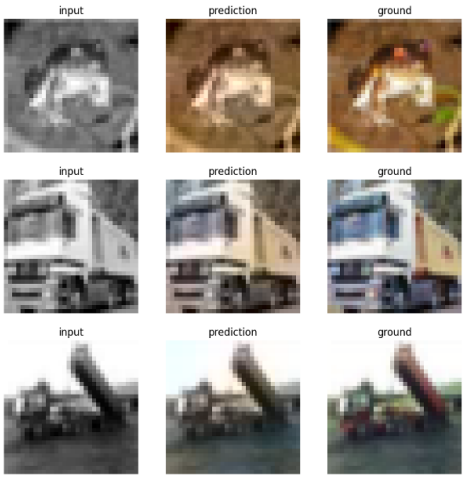


**Figure 14: Cifar Base Images**

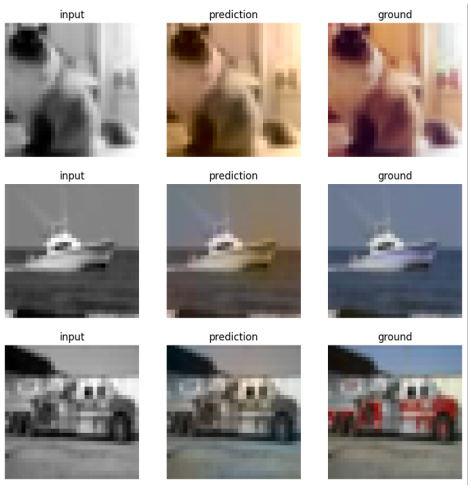


**Figure 15: Cifar Base Imges**

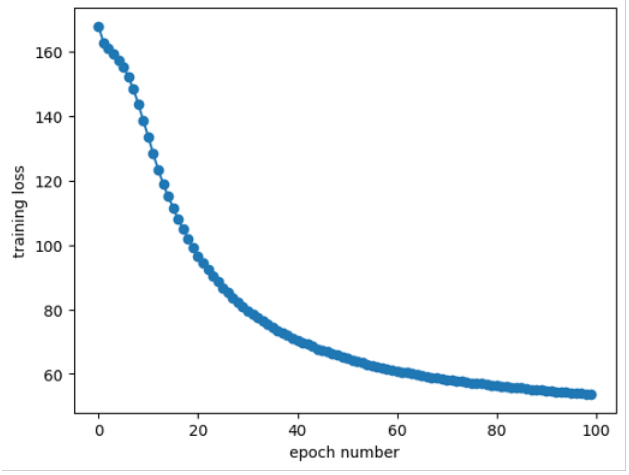


**Figure 16: Training Loss Plot**

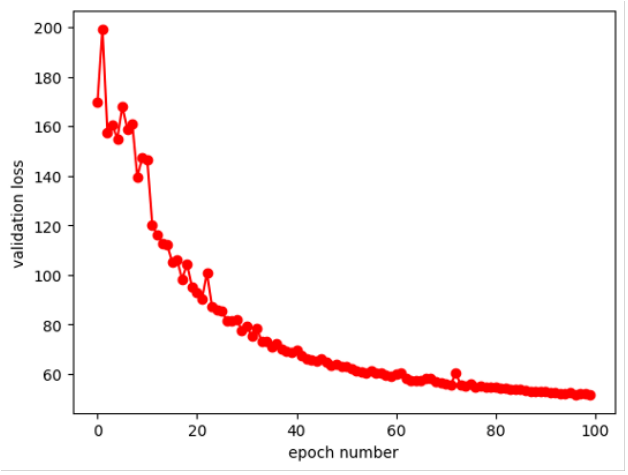


**Figure 17: Validation Loss Plot**

The model trains pretty well on CIFAR-10, and produces decent output pictures. These predictions are not very accurate most of the time, but they show that the model has potential. We had thought that the model would work better on CIFAR-10, but overall, we did not see a significant difference in the results of our model on the CIFAR-10 dataset when compared to the MIRFLICKR dataset.

### 3.1. SGD vs Adam

Previously we trained our model on the MIRFLICKR-25000 images and included the hyper-parameters seen in Table 3, most notably the SGD optimizer. The output can be seen in Figure 18.

| Learning Rate | Epochs | Optimizer |
|---|---|---|
| 0.0001 | 100 | SGD |

**Table 3: Model Hyperparameters using SGD**

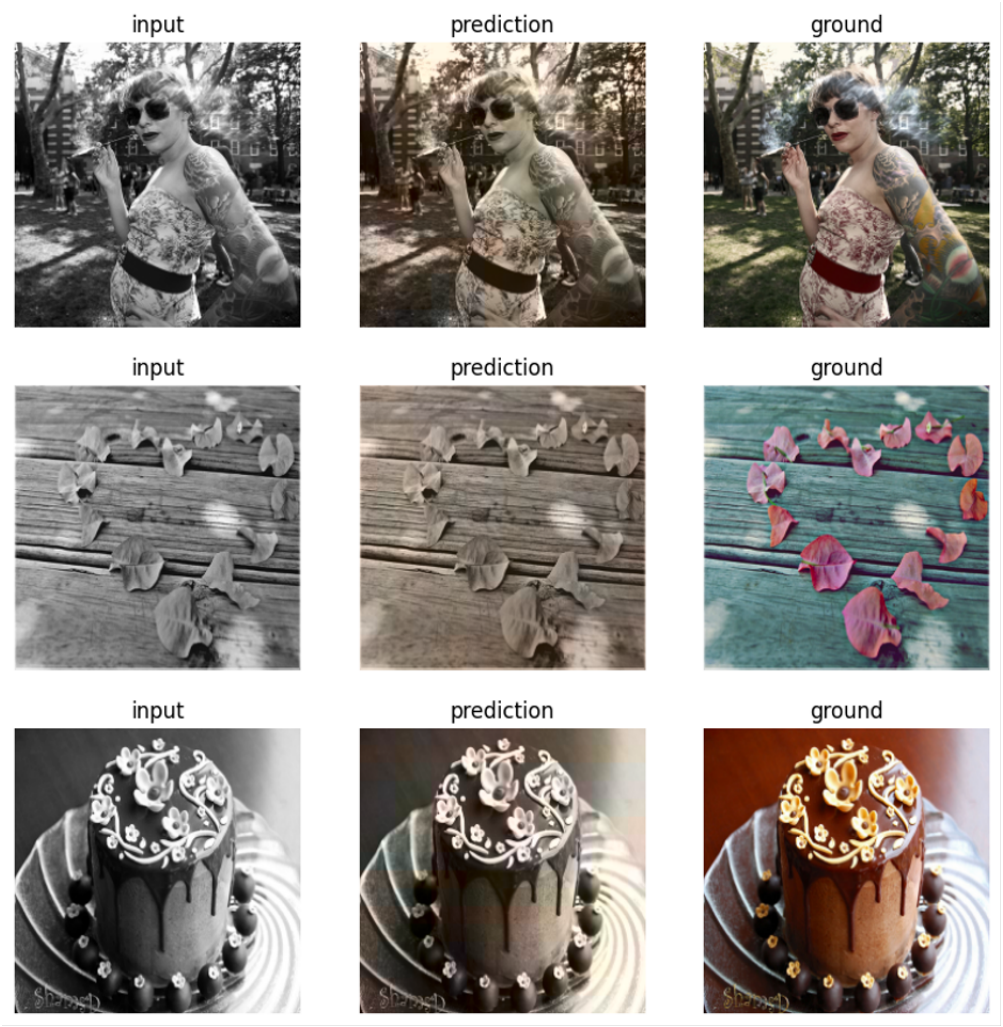


**Figure 18: Base Images using SGD**

The loss plots for the training and validation can be seen in Figure 19 and Figure 20 respectively.

We ran our best-performing model with SGD (instead of Adam) as the optimizer but with 100 epochs to see the results. As seen between Figure 18 and Figure 13, there is a clear difference between using the Adam optimizer and the SGD optimizer. The predicted images don't show much color at all, and instead, they look biased in the blue-yellow color space (since they are colored a

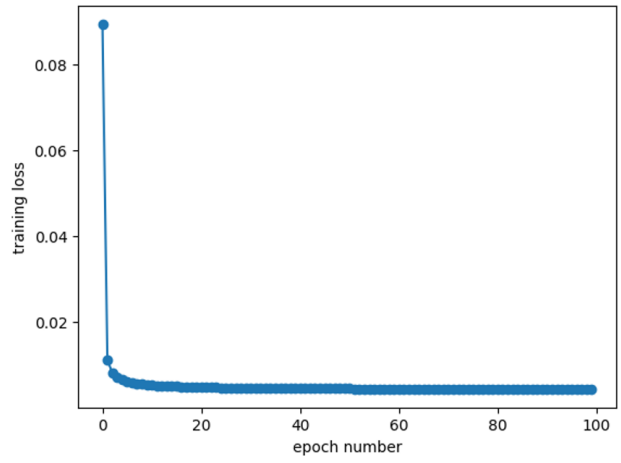


**Figure 19: Training Loss Plot**

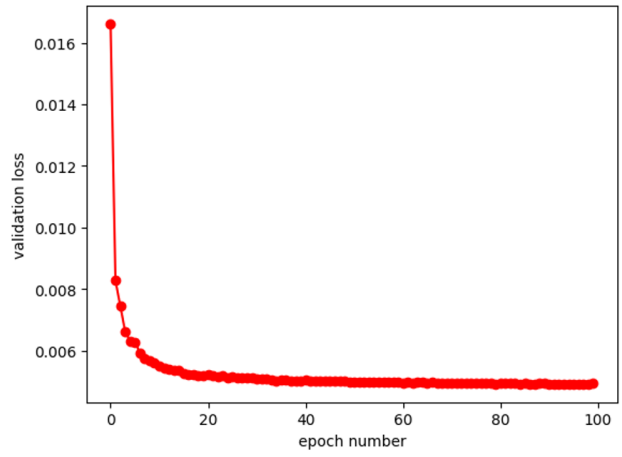


**Figure 20: Validation Loss Plot**

weak yellow). In the loss plots between Figure 11 and Figure 12 versus Figure 19 and Figure 20, the Adam optimizer tends to converge faster. Overall, the Adam optimizer performs significantly better than the SGD optimizer.

# 4. Summary

## 4.1. Discussion

Previous works had approached image colorization as either a regression, or a classification problem. For regression, we decided that using MSE loss was the easiest to work with. For classification we had tried Cross Entropy loss, but could not produce viable results. In the future, it might be worth looking into more complex variations of classifications, as seen in Zhang et. al [4]. Our CNN was similar in terms of base model structure as previous implementations, but we found that using MSE loss and the Adam optimizer produced the best results for image colorization with only a CNN.

## 4.2. Qualitative Limitations

We can across a number of issues during this project. One glaring issue with our model is when the ground truth image is also grayscale. Our model could not determine that the ground truth output was also a grayscale image. As a result, our model would attempt to add color to a

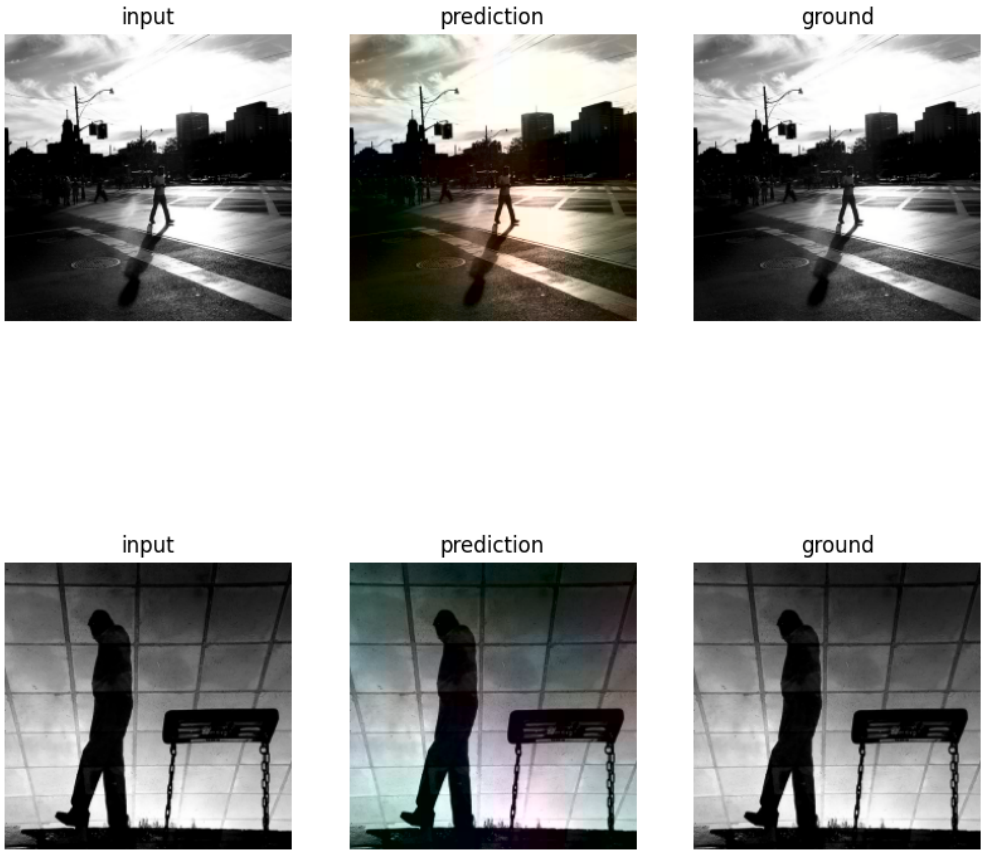


**Figure 21: Output with grayscale ground truth images**

completely grayscale image as seen in Figure 21. This is very consistent, and we did not find a good solution to this problem.

Another problem with our model was blocky colorization. As seen in Figure 22 and Figure 23, the model tends to color the input images in blocks. It is more apparent in Figure 23, because the model that outputted that image had a max of only 256 features. It is still noticeable in Figure 22, but the blocks of color are smaller when there are a max of 512 features. It is hard to see in our final model outputs, but it is still there. We suspect that it has to do with normalization or upsampling during the forward feed of the CNN. We could not find a solution to this problem.

### 4.3. Technical Limitations

One challenge was on the technical side. We noticed early on that memory usage was going to hinder our ability to train on a large data set. We initially utilized Google Colab which had a


**Figure 22: Cifar Base Images**


**Figure 23: Cifar Base Imges**

maximum of 15 GBs. However, using the entire data set from the MIRFLICKR-25000 (released in 2008) could not be supported on Google Colab's base memory plan. We then proceeded to utilize Princeton Adroit but that also took longer than expected. From there, we then purchased the pro version of Google Colab, which increased memory usage from 15 GB to 25 GB, which was sufficient to train the MIRFLICKR-25000 and CIFAR10 data set.

### 4.4. Future Work

We wanted to try and implement a Generative Adversarial Network for image colorization, but we did not have time to explore this option. Previously, we mentioned that Zhang et al. 2017 had used a more complex classification objective loss function (multinomial classification with class rebalancing) [4]. We also wanted to try implementing something similar but again, we did not have time. We could have also compared our model with existing implementations, but we had thought that our model is not good enough for a fair comparison. Although we tried our model on two different datasets, it might be worth trying to use more data and also maybe a less diverse dataset. We could have also tested our model on human subjects by making them assess the naturalness of our output.

## 5. Acknowledgements

First and foremost, we would like to offer our deepest thanks to our project mentor Maxine Perroni-Schar for their continued support and for their active role in advising this project (especially when we, unfortunately, had a project member leave due to personal reasons). Their advice and guidance during the weekly meetings were invaluable and contributed greatly to this project's success.

## 6. Honor Code

This paper represents my own work in accordance with University regulations. /s/ Gedeon Guercin /s/ Erkhem Unenbat.

## References

[1] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, vol. 35, no. 4, pp. 110:1–110:11, 2016.

[2] P. Isola *et al.*, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.07004, 2016. Available: http://arxiv.org/abs/1611.07004

[3] L. Melas-Kyriazi, "Image Colorization with Convolutional Neural Networks," *Git)*, 5 2018. Available: https://lukemelas.github.io/image-colorization.html

[4] R. Zhang *et al.*, "Real-time user-guided image colorization with learned deep priors," *ACM Transactions on Graphics (TOG)*, vol. 9, no. 4, 2017.