Automated Colorization Of Grayscale Images Using Deep Learning Techniques

Revathi Bhuvaneswari rbhuvaneswari@fordham.edu
Tianying Luo tluo9@fordham.edu
Yue Zeng yzeng44@fordham.edu

Abstract

We present three different approaches for colorization of grayscale images using deep learning. Analysis on favorable and unfavorable candidates for image colorization are explored with experiments on diverse datasets. A baseline CNN model is used to benchmark the process. A pure Convolutional Autoencoder explores how applications of image denoising can be applied to colorize input grayscale images. Finally, two different transfer learning approaches are presented with VGG16 and Inception-ResNet-v2 models in order to extract finer image features and produce realistic color predictions.

1 Introduction

Colorization has wide range of useful applications in restoring historical black-and-white photographs, MRI scans and even analyzing satellite images. This colorization process is however tedious to do by hand. In the past few years there has been a rise in the research for automating the colorization process in the Computer Vision industry and notable findings have been published. Recently, automated colorization has even paved way to video frames colorization.

Figure 1. Grayscale Image Colorization

Our paper focuses on colorizing grayscale images using deep learning techniques. The process of automated colorization of images is highly challenging because the different color combinations that can be applied to a single object in an image. In order to extract meaningful features from the images, Convolutional Neural Networks are used as a base in all our models. The prediction of pixel values can be done through a classification or regression approach. In order to allow room for experiments with different model architectures and image types, we extensively used the regression approach for our training process.

2 Dataset

Colorization data is everywhere as we can extract the grayscale channel of any colored image. It is crucial to have a large diverse set of images during training in order for the model to learn realistic color information. We mainly used the Visual Genome image dataset, which has 108, 077 images with different subject, dimensions and color quality. For this project, we used a subset of 15, 000 images to train the final model, and sampled smaller sets for hyperparameter tuning. All images were resized to either 256 x 256 or 224 x 224 for computational efficiency. Out of the 15, 000 images, 20% was held out for validation and 10% was held out for training.

In addition to such a diverse dataset, we also used smaller datasets that have specific classes in order to measure the ability of our model to produce realistic colorization on unseen data and get an idea of what kind of images are easy/difficult to colorize. The McGill dataset that has images of different classes such as Nature, Man-Made, Fruits, Animals etc. was heavily used in the beginning of our training experiments. Additionally, the Oxford Flowers dataset was also used. Our final dataset was a total of 16, 590 images which is a combination of the three main datasets.

Figure 2. Final Dataset Preview

3 Approach

We took a regression approach to predict image pixels for the colorization and experimented with three main model architectures:

(1) Baseline CNN (2) Baseline Autoencoder (3) Transfer Learning with Pre-Trained CNNs

The implementation details along with experiments on these three models are discussed in the subsequent sections. Additionally, we also experimented with different image preprocessing techniques, loss functions and overall evaluation metrics for our models.

3.1 Color Space and Preprocessing

In order to simplify the training process, we converted our images from the default RGB colorspace to the CIE L*a*b* colorspace as a preprocessing step. L* represents the lightness in images with values ranging between [0, 100] (black to white), a* represents the green-red spectrum [-128, 127] and b* represents the blue-yellow spectrum [-128, 127]. Since all the color information of an image is present within the a*b* channels, our prediction task becomes easier. Additionally, we can use the L* channel as grayscale input for our models. The final colorized image can be obtained by concatenating the input L* channel with predicted a*b* channels. The initial RGB image of 3 channels is scaled to values within [0, 1] by dividing with 255. In cases where the LAB color space was used, each individual channel's values are scaled to [-1, 1] to make it zero-centered. Additionally, random image augmentations such as random rotation, flips and zooms were performed in order to make the training process more robust.

3.3 Loss Functions and Evaluation Metrics

The choice of loss function during training drastically affects the quality of colorization in the output, especially for a regression approach. We experimented with the following loss functions.

L2 Loss The L2 or MSE loss function minimizes the squared differences between the true and predicted values, which in our case are the a*b* pixel values.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Due to the high penalization of incorrect predictions, L2 loss helps generate an output that resembles the ground truth as close as possible. However, this could be a downside for colorizing objects that can take on multiple color values. For example, a shirt that can be 'Red' or 'Blue' in reality could end up being colored in a neural sepia tone by the model in its attempt to reduce the loss associated with picking one color over another.

L1 Loss The L1 or MAE loss function, on the other hand, minimizes the absolute differences between the true and predicted a*b* pixel values.

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

This loss is more robust to outliers compared to L2 loss function. We believe L1 can help prevent the sepia coloring of output images because of less penalization, though there are chances of the model failing to learn the finer details of colorization.

In addition to using these loss functions to quantify the performance of our models, we also used 'human evaluation' to measure the quality of colorized output images since this task is different from a traditional regression problem. Performance statistics are provided in the below sections.

4 Baseline CNN

Convolutional Neural Network takes a grayscale image as input and generates a colorized version of the image as its output. The basic idea is that use one channel to predict the other two channels, then concat them into three channels. Figure 3 shows an example of such a pair of input and output images which are in our ideal state.

Figure 3. Left - Original(output), Right - Grayscale(input)

In this part, we design and build different simple and basic architectures. As a baseline model, we adjust different parameters to generate better results. Most of current research have utilized this baseline model to add colors on the images. Some of them combine pre-trained model with basic CNN architectures. However, in this part, we just work on building a basic model through pure parameters without any other layers. Even though, there are some limitations of this model, the goal of this part is to see the best that model can produce. Additionally, compare it with other models to get better results in the future work.

4.1 Implementation

The core logic is to convert the input image which have 3 channels to LAB color space. Treat the converted image which only have one color or one L channel as model's input. The other A and B channels are extracted as the target values.

During the training time, the images which are read all have 3 channels. The channels are corresponding to red, green and blue of RGB color space. The shape of the pixel dimension is (224, 224). Then convert the images into black and white. In other words, the images will have only one channel, luminance (L). We would use the images which have L channel as the input of model to predict the other channels.

During the testing time or the modeling time, the input of the model are black and white images (L channel). Also, the shape of the images are (224, 224, 1). It generates two arrays which are corresponding to the A and B channels of the LAB color space. Then these three channels are concatenated together as the predicted image output.

For our networks, we create filters to link the input and output, the number of the filters connected with the complexity and the number of parameters in the convolutional neural networks. When we calculate the value through the convolution, we used batch norm (BN) instead of bias terms behind every convolution before we use activation function. Additionally, we used ReLUs as activation functions throughout except at the last to output AB channels - before we used simple calculation to scale the values between -1 and 1. At the end of the layer, we use a Tanh activation function to predict values. According to the function of Tanh, for any value you give the Tanh function, it will return -1 to 1. Besides, for the last two layers, we need to use upsampling layers or transpose layers to double the size of the image.

Input: 224 x 224 x 1	
3 x 3 Conv, 64, S = 2	224x 224x 64
3 x 3 Conv, 64, S = 2	112x 112 x 64
3 x 3 Conv, 128, S = 2	112x112 x128
3 x 3 Conv, 128, S = 2	56 x 56 x128
3 x 3 Conv, 256, S = 2	56 x 56 x256
3 x 3 Conv, 256, S = 2	28 x 28 x 256
3 x 3 Conv, 512, S = 2	28 x28x1512
3 x 3 Conv, 512, S = 2	14x214x 512
3 x 3 Conv, 512, S = 2	28 x28x512
3 x 3 Conv, 256, S = 2	56 x56x256
3 x 3 Conv, 128, S = 2	112x112x 128
3 x 3 Conv, 64, S = 2	224 x224x 64
3 x 3 Conv, 256, S = 2	224 x224x2
Output: 224 x 224 x 2	

Figure 4. The architecture of our networks

4.2 Experiments

In order to tune hyperparameters for the model, a smaller subset of the dataset of around 1000 images was used for training with 20% for validation and 10% for testing. For different CNN architectures, we thought to adjust different dropout, kernel_regularizer, batchNormalization and epochs to train each of our basic model in order to generate best results.

Firstly, we trained the model without BatchNormalization, Dropout, kernel_regularizer and weight initialization. After 100 epochs, the model produced brownish images, as shown in Figure 4. Then, we added BatchNormalization and used the same epochs, but they still produced one color, the same as before. When we printed the arrays what we predicted, we found that the values are so nearly and similar. So the picture actually have three main colors, based on yellow-brown range.

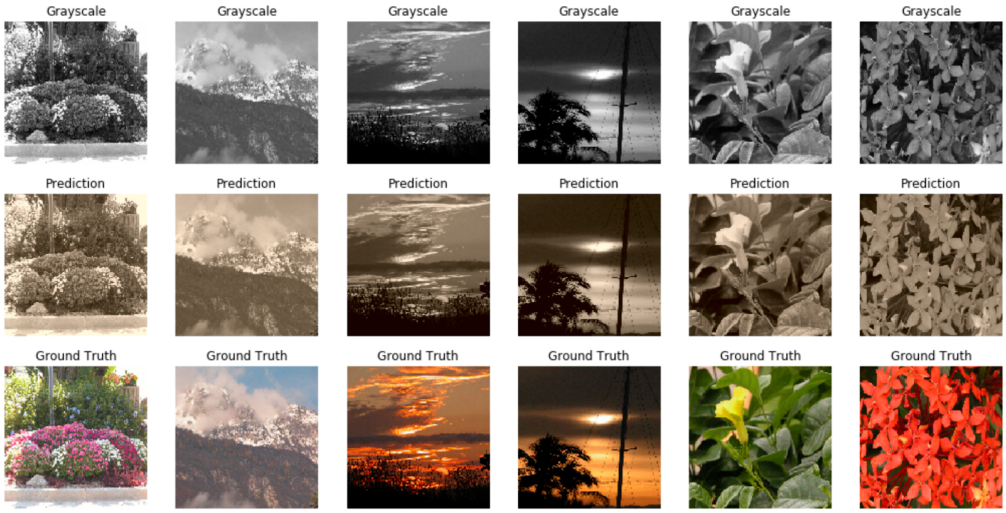

Figure 5. The prediction of CNN without BN, dropout and kernel_regularizer

After that, we added kernel_regularizer, especially, the parameter of kernel_regularizer was set to L2 regularization. However, we deleted dropout layers. With 100 epochs, they generated not all of the brownish images, some of them have some green color on the image. Compared with original images, because of the nature dataset, some of the green parts are still on the object.

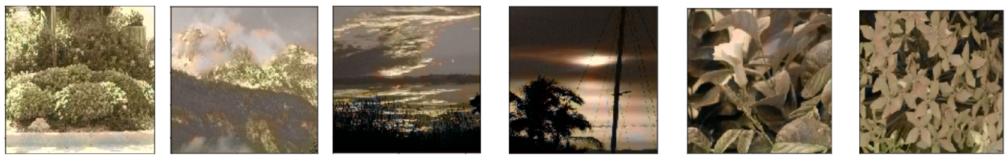

Figure 6. The prediction of CNN without dropout and with BN, kernel_regularizer

After the image generated some green and yellow-brown color, to make better results, we added more layers and more parameters. This time, we removed kernel_regularizer, dropout, and added transpose layers. For these kind of layers, they usually arise from the desire to use a transformation going in the opposite direction of a normal convolution. After 500 epochs, we got a better result, as shown in Figure 7, and the images look colorful. Besides, compared with original image, all of the colored part has been on the objects with no color leakage problems. It also looks meaningful. Considering the model architecture and training time, we updated the epochs from 500 to 1000. The generated images all look more colorful and meaningful than with 500 epochs.

Figure 7. The prediction of CNN with BN, transpose layers, 500 epochs

Figure 8. The prediction of CNN with BN, Transpose layers, 1000 epochs

4.3 Results

The final model with the above discussed parameters was trained with two different combinations of dataset - a smaller subset of 1,590 images belonging to Nature / Flowers class, and a larger final dataset of 16,590 images. Each dataset used a 10% split for validation and 20% split for testing. While the dimension of input images is (224, 224, 3). Because of the size of the dataset and the training time, we had to do them in batches due to constraints. Table1 shows the performance of baseline_CNN Model. Figure 9 shows the result of the final model.

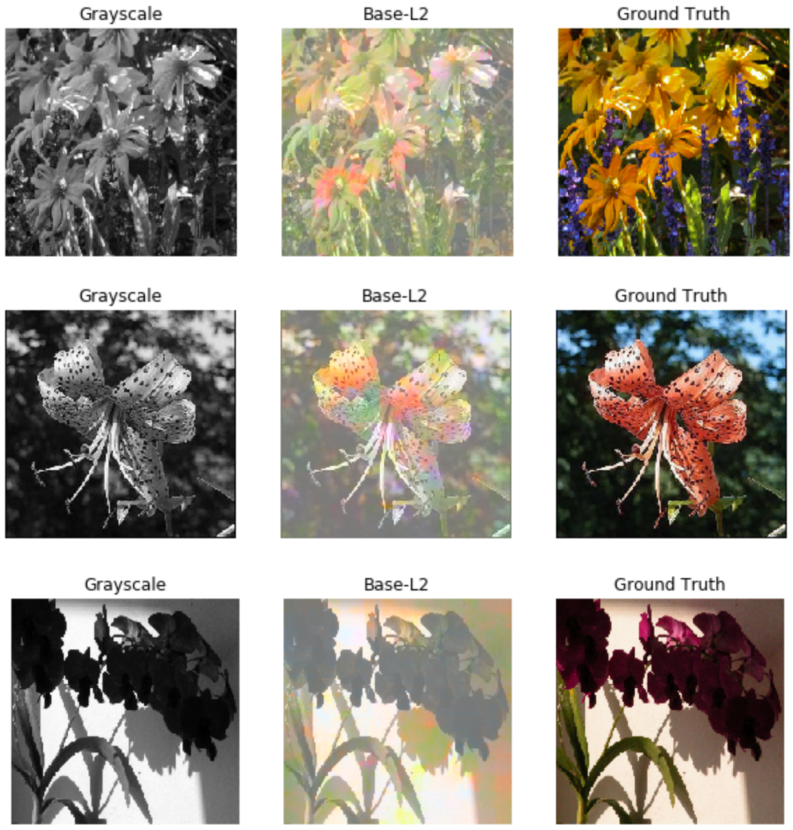

Figure 9. Final Model training results

Table 1. Performance of Baseline_CNN Model

Dataset	L2 Loss			Mean Absolute Error			Epochs
	TRAIN	VAL	TEST	TRAIN	VAL	TEST	
Nature	0.0010	0.0034	0.0020	0.0456	0.088	0.0786	1000
Final 10K	0.0090	0.0101	0.0193	0.0796	0.0893	0.0803	500

5 Baseline Autoencoder

The Colorization Autoencoder takes grayscale images as input and colored images as output. It can be treated as the opposite of denoising autoencoder. Instead of removing noise, Colorization Autoencoder adds noise/color to grayscale images. Two methods are implemented to experiment with colorization task, one using RGB color space; the other using LAB color space.

Due to the autoencoder's ability, it can learn to represent the given grayscale image into a latent (compressed) form, which is done by the encoder. After the image has been brought down to its latent form, the decoder will reconstruct the coloured image using that latent representation. During the colorization process, autoencoder will extract spatial features which may be responsible for the target image, and store the knowledge required to color an image by learning features from grayscale images. With enough training samples, it will tend to know which region will be colored with which color.

5.1 Implementation

An autoencoder network consists of 2 sections: i) Encoder and ii) Decoder. Our target is to construct a RGB form of image from a grayscale image. Here we experiment with two methods.

Colorization Autoencoder using RGB: In this simple model, the encoder networks consists of 4 convolutional block and same the decoder. Output for every ConvNet layer will be activated by "ReLu" function and the final output layer by "tanh" function, which rescale the range into (-1, 1). Here we applied mean pixel squared error to determine the loss. The difference of pixel values between the original image and reconstructed one is measured by Mean Squared Error during the training process. MAE and MSE were both experimented during the training. But MSE as a loss function would give better outputs.

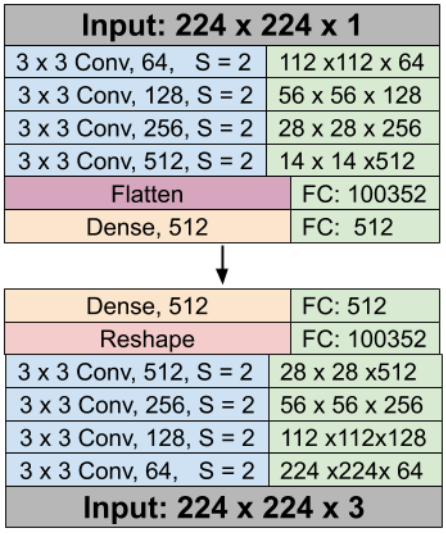

Figure 10. Architecture of AE-RGB- Model

Colorization Autoencoder using LAB: For AE-LAB-Model, we implemented eight ConvNet blocks to encoder and another eight ConvNet and UpSampling layers to predict ab layers in L*a*b colorspace. **Encoder** is consisted of eight convolutional layers with activation function ReLu and strides 3 in order to decrease the width and height of the latent space vector. **Decoder** consists of convolutional layers with upsampling layers to restore the dimensions of the original input image. For the last layer, 2 filters represent the ab channels. We used tanh activation function instead of ReLu because we normalized the ab values between (-1, 1) and tanh function is used to squash values into (-1, 1). The resconstructed images will be concatenated by the predicted ab layer and lightness layer. Here, Keras callbacks API is used to specify "loss" (mean pixel squared errors of the ab layer between the original image and the predicted ab layer) to monitor on training and validation dataset. An improvement of minimizing the MSE score will be specified, too. In this model, learning rate scheduler is also used to monitor the "loss" as well as if there's an improvement for a "patience" number of epochs. Once the learning ability stagnates, the learning rate will be reduced by the factor of 2-10.

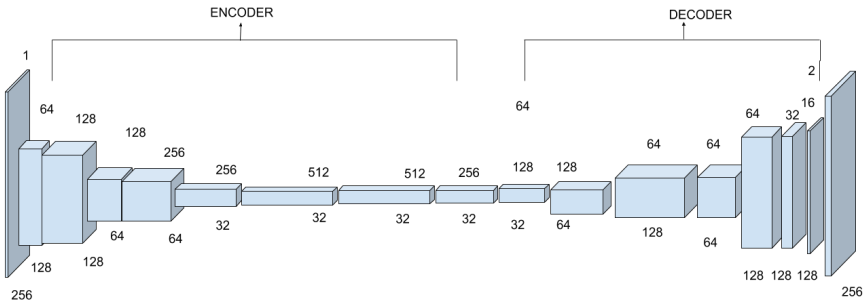

Figure 11. Architecture of AE-LAB- Model

5.2 Experiments

Colorization Autoencoder using RGB:

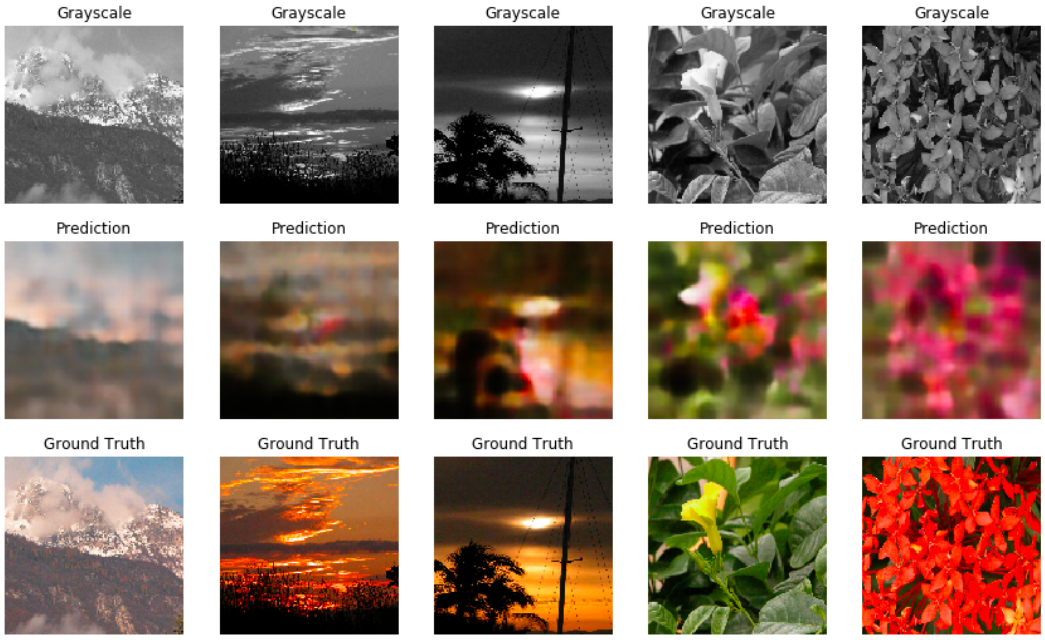

Figure 12. The result of the colorization ae using RGB

Colorization Autoencoder using LAB:

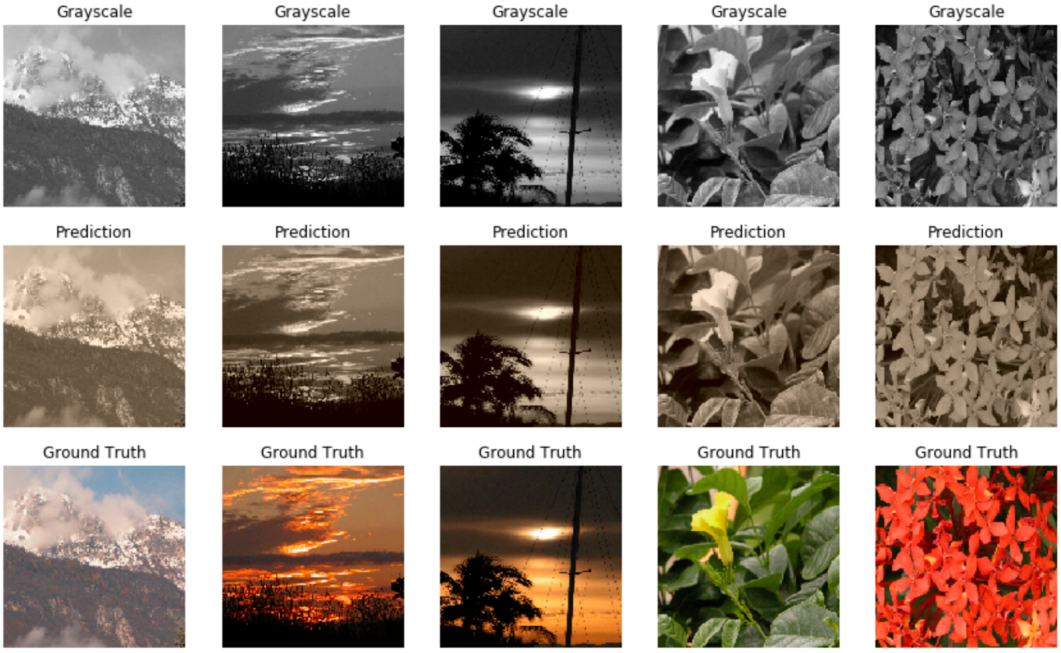

Figure 13. The result of the colorization ae using LAB

5.3 Results

The baseline AE model using RGB could depict the colors very well. However, due to the limitations of autoencoder, it hardly reconstruct the image to a high-resolution level. For The baseline AE model using LAB color space, since the work has minimized to only predicting the ab layer, and grayscale images have been able to remain. The network is able to learn the basic colorization based on the lightness layer despite giving out the more brownish color. That's because brown is a very average color so the network tries to minimize the deviations from expected pixel value to cover the original image with sepia effect. The takeaway from experimenting with two basic AE models is that feature extraction matters a lot in training a successful colorization autoencoder model. Next, transfer learning technique will be introduced and used in finalizing our model.

6 Transfer Learning with Pre-Trained CNNs

The image colorization problem could benefit from a large and diverse dataset when fighting against de-saturated predictions with neural sepia tones. We decided to take advantage of models that were already trained on the ImageNet dataset in order to better extract our datasets' features and in turn improve overall output quality.

These pre-trained models could aid with learning about the relations between different objects in an image, thus helping produce more realistic predictions. We experimented with the VGG16 and the Inception-ResNet-v2 models, both of which achieved top accuracies in the ImageNet image classification challenge. An Encoding-Decoding CNN architecture was used for experiments with these two model integrations.

6.1 VGG16

6.1.1 Implementation

Instead of training the network from scratch, VGG16 pretrained model will be used an encoder.

VGG16 is a vision model consisted of CNN architecture trained to classify between 1000 class of ImageNet dataset. By using the transfer learning technique, VGG16 model will be reused as the starting point for our feature extraction task as an encoder and help train our network more efficient and generate more faster.

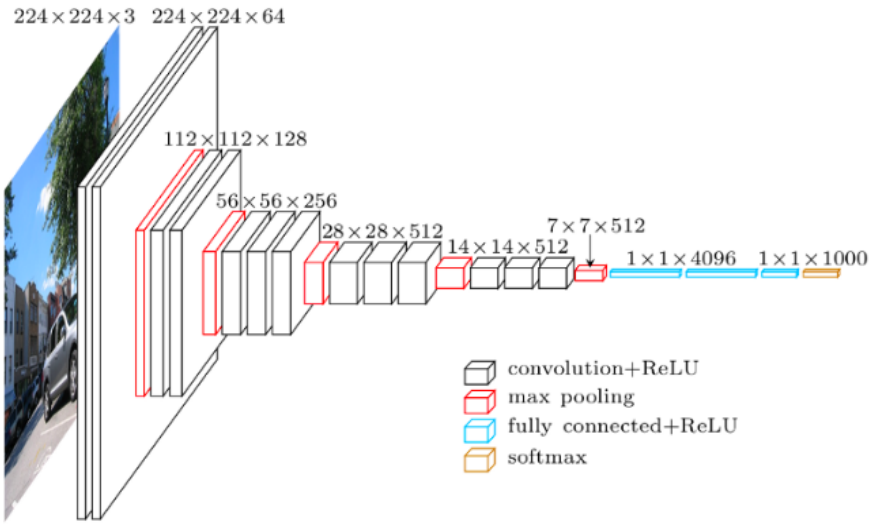

Figure 14. Architecture of Vgg-16 Pretrained Model

VGG16 Encoder The above figures illustrates the architecture of VGG16: the input layer takes an image in the shape of (224, 224, 3) and the output layer is a softmax prediction of 1000 classes. The first 18 layers up until the last max pooling layer in the shape of (7 x 7 x 512) is regarded as the feature extraction part, and the next fully connected layers up until the softmax layer in the network are considered the classification part of the model. The VGG16 network layers will be iterated to the 19th layer with the dimension (7, 7, 512). Such latent space volume will be regarded as a feature vector to be the input to the decoder.

Custom Upsampling Decoder The decoder takes the feature vector as input in the shape of (7, 7, 512), and passes through convolutional and upsampling layers. The last upsampling layer will produce predicted ab layer in the shape of (224, 224, 2). The number of total trained weights is around 14M.

ENCODER	**Input: 7 x 7 x 512**	
	Output: 7 x 7 x 512	
DECODER	**Input: 7 x 7 x 512**	
	3 x 3 Conv, 256	7 x 7 x 256
	3 x 3 Conv, 128	7 x 7 x 128
	2 x 2, Upsampling	14 x 14 x 128
	3 x 3 Conv, 64	14 x 14 x 64
	2 x 2, Upsampling	28 x 28 x 64
	3 x 3 Conv, 32	28 x 28 x 32
	2 x 2, Upsampling	56 x 56 x 32
	3 x 3 Conv, 64, S = 1	56 x 56 x 16
	2 x 2, Upsampling	112 x 112 x 16
	3 x 3 Conv, 32, S = 1	112 x 112 x 2
	2 x 2, Upsampling	224 x 224 x 2

Figure 15. Architecture of Vgg16-AE Transfer Learning Model

6.1.2 Experiments

Different classes are experimented with Vgg16-AE before we move on to training a larger dataset.

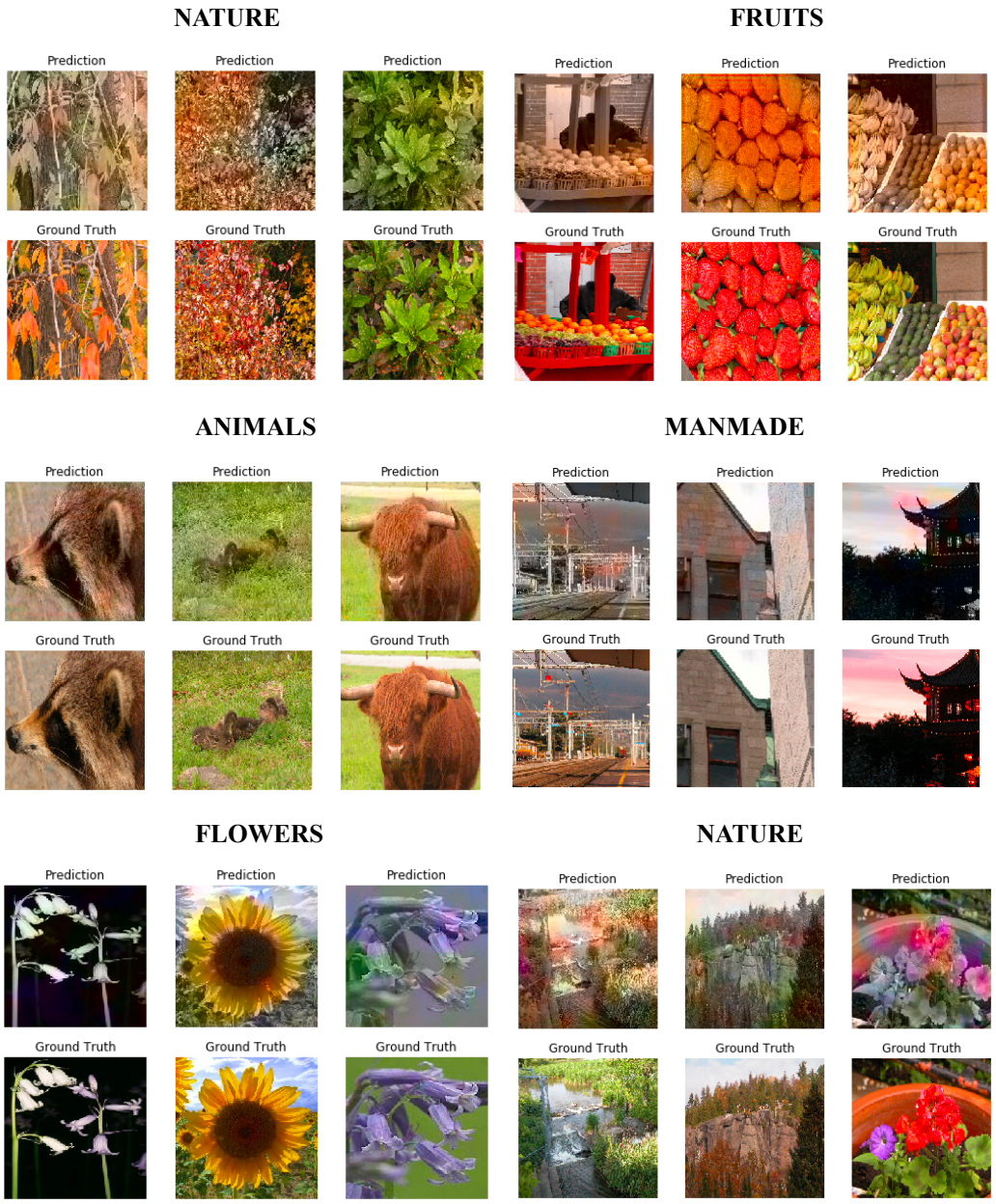

Figure 16. Different classes are experimented with Vgg16-AE

Each dataset differs from another, how well the network can predict the color depends on the type of dataset. In general, with enough data and trial times, the model will be able to extract signature feature and map to the right color. For example, in the class of flowers and animals, the prediction is almost similar to the ground truth. However, when the details become rich and complicated, the color tone tend to become brown or even fails to predict. In order to improve the model. The larger dataset and minor adjustment of batch normalization will be added.

6.1.3 Results

When the dataset increases to 10K, the colorization on the images appears very vague. Thus, we increased the epochs to 20K in order for it to learn better, however the outputs seems colorful and

not exactly map the right color to the according feature. Due to the RAM constraints, the largest dataset we've been able to train is 16K, and after 500 epochs, the effects have been improved but not ideal. In the following step, we'll experiment with Inception-ResNet-v2 model, which has a much deeper learning architecture and better classification accuracy.

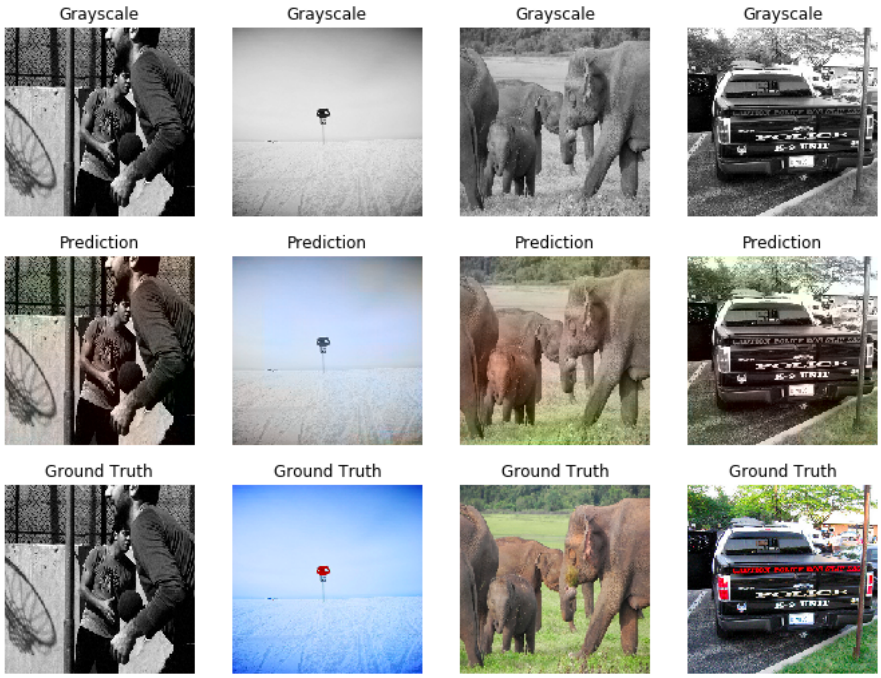

Figure 16. 10k dataset, with Vgg16-AE, with 100 epochs

Table 2. Performance of Vgg16-AE Transfer Learning Model

Dataset	L2 Loss			Mean Absolute Error			Epochs
	TRAIN	VAL	TEST	TRAIN	VAL	TEST	
Final 10K	0.0094	0.0166	0.02	0.0376	0.078	0.0706	100
Final 16K	0.0076	0.0113	0.0117	0.0586	0.0706	0.0754	486

6.2 Inception-ResNet-v2

The Inception-ResNet-v2 model is the latest available version of Google's Inception series models in combination with Microsoft's ResNet architecture. It is much deeper than VGG16 but is computationally efficient with less parameters comparatively. This model also has achieved higher accuracy in the ImageNet image classification challenge.

6.2.1 Implementation

Transfer learning from the Inception-ResNet-v2 model is done by integrating its output with custom feature extracting convolutional layers before proceeding to the decoding stage. The model has two input branches and requires different processing of the input L* channeled image. The structure of the entire model gives it the capability to accept input image of any H x W dimension, with the only requirement being that it should have 1 channel. For baseline and parameter tuning image dimensions of (256, 256, 1) were used. However, for the final model of nearly 17K images, the images were resized to (224, 224, 1). For demonstration purposes, an input image shape of (256, 256, 1) is used to describe the model architecture in this section.

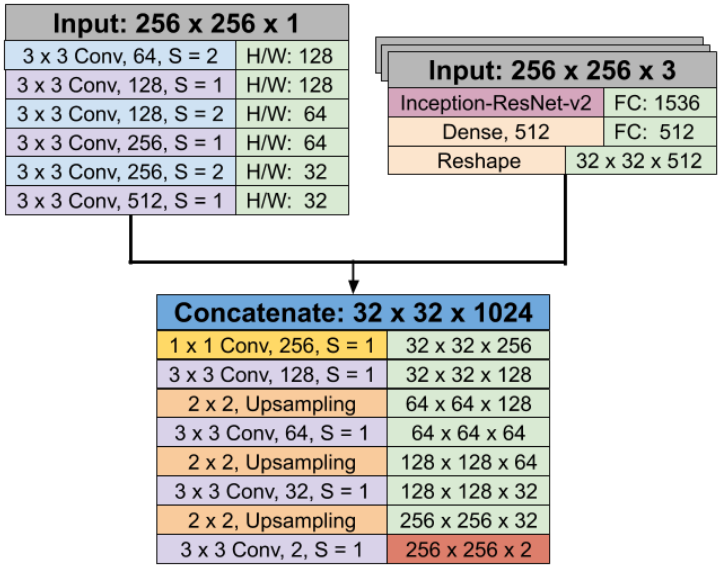

Figure 17. Architecture of Inception-ResNet-v2 Transfer Learning Model

Custom Convolutional Layers This part of the encoding stage is comprised of a series of convolution layers of kernel size 3 x 3 that takes in an image of size (256, 256, 1). Convolution with stride 2 is used instead of Maxpooling in order to learn during the downsampling process and improve the model's extracting ability. Alternating Conv. layers with stride 2 and 1 are applied.

Inception-ResNet-v2 Layers The In-ResNet model comprises of the top-level feature extracting Conv. layers and classifying fully-connected dense layers. For this task, only the Conv. layers are used to reduce overall complexity and have flexibility in input dimensions. The model, however, expects its input to have 3 channels, so the original L* input of size (256, 256, 1) is stacked three times to produce a new image of (256, 256, 3). A Global Average Maxpooling layer along with a Dense layer is added at the end of the In-ResNet model to get a 2D tensor instead of a 3D one and control the number of parameters for the model. This extracted feature information is reshaped, concatenated with the output of custom encoding layers and passed through a Conv. layer of size 1 x 1 with stride 1 for feature pooling before the next stage.

Upsampling Decoding Layers The input for this decoding stage is of shape (32, 32, 256), which is passed through alternate Conv. and Upsampling layers in order to get the output image. The final Conv. layer with 2 kernels helps produce a prediction of shape (256, 256, 2).

All layers in the model use 'ReLU' activation, except the last layer which uses 'Tanh'. Each Conv. layer is also followed by a BatchNormalization and Dropout layer, which were experimented with during the training process. The total number of trainable parameters for the model is around 3M.

6.2.2 Experiments

In order to tune hyperparameters for the model, a smaller subset of the dataset of around 1000 images was used for training with 20% for validation and 10% for testing. The model with weight initialization using He Normal along with BatchNormalization layers produced better results than baseline. While the L1 loss model produced sharper images, the L2 model was able to learn a wide spectrum of colors quickly and predict realistic outputs as a result. Dropout layers with a dropout rate of {0.1, 0.3, 0.5} were tried to help with issues of overfitting. However, adding a dropout produced images with a neutral sepia tone and so the dropout layers were removed. We also experimented with different batch size + epochs combination and found that a batch size of 100 worked best for the model with input shape of (256, 256, 1). Additionally, at least 100 epochs were required to produce meaningful results, and maximum of upto 500 epochs were tried.

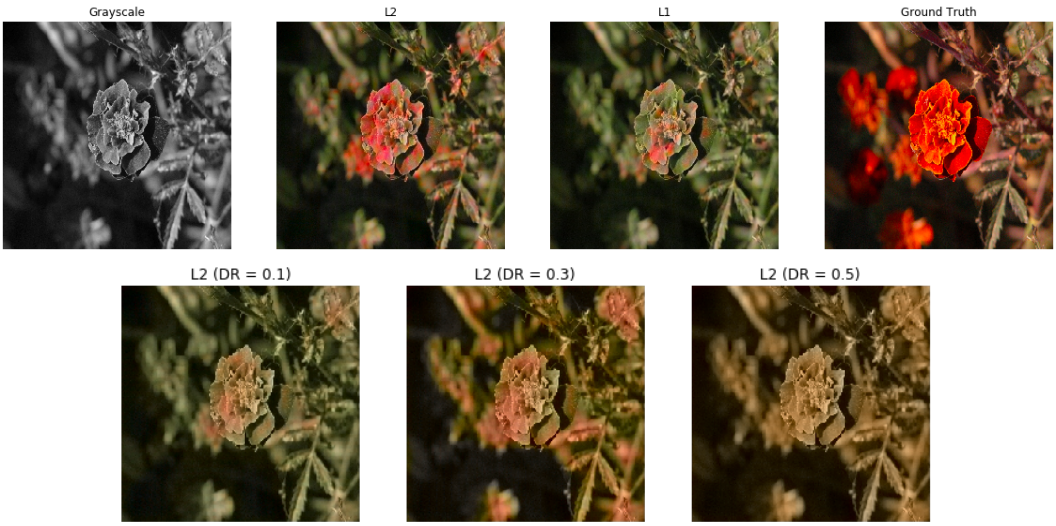

Figure 18. Performance of BatchNorm Model for Different Loss Functions and Dropout Rates

Special care was taken during tuning of the optimizer and respective learning rate due to its significant impact in the training process. Adam optimizer was used for training because of its robustness, and LearningRateScheduler along with ReduceLROnPlateau were implemented to try out different learning rates. The default Adam learning rate of 0.001 with $\beta_1 = 0.9$, $\beta_2 = 0.999$ gave good results compared to learning rate in {0.1, 0.01}. Reducing the initial learning rate by a factor of 0.1 when the loss plateaued helped speed-up the learning process. L2 kernel regularization was also tried with {0.01, 0.001} to fight the effects of overfitting. However, the results ended up being grayscale or neutral sepia toned so this was not included.

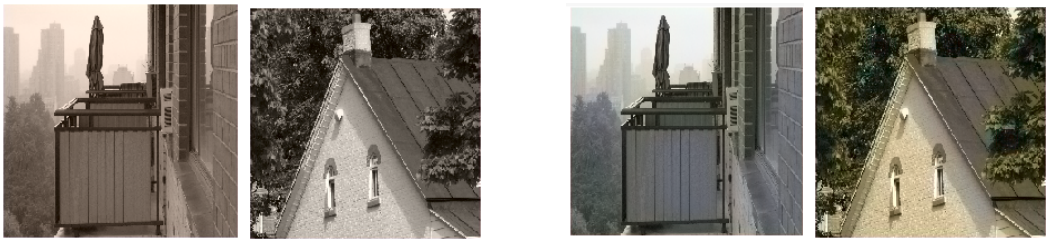

Figure 19. Left - With L2 Regularization (0.01), Right - Without L2 Regularization

6.2.3 Results

The final model with the above discussed parameters was trained with two different combinations of dataset - a smaller subject of 1, 590 images belonging to Nature / Flowers class, and a larger final dataset of 16, 590 images. Each dataset used a 10% split for validation, 20% split for testing. While the smaller subset used images with dimension (256, 256, 3), the larger one used (224, 224, 3) in order to speed-up the training process. This section provides metrics of loss functions along with sample predicted images in the test set.

Training on the Nature / Flowers datasets produced very realistic looking images in just 100 epochs for the unseen test split of data. Providing a diverse set of images for this particular class during training helped the model learn acceptable color combinations for each image. For example, sunflowers should be yellow and stems should be green. Below figure displays the predicted images on test set. These results correlate closely with real nature color spectrum.

Figure 20. Nature Success Case Predictions

There were few failure cases with this dataset as well. Images where neutral tones like brown or white are predominant with only a few colorful regions produced images where these smaller portions remained uncolored while the rest of the image got colored.

Figure 21. Nature Failure Cases, Top - Predictions, Bottom - Ground Truth

The final 16.5K dataset was initially trained as a whole with appropriate train/val/test splits of images in dimension (224, 224, 3). However, due to the complexity of the model and the size of the dataset the model was only able to train for 107 epochs, with each epoch taking on average 330 seconds (5 minutes), with a total runtime of about 9 hours. The results with this run was not favorable with color leakage problems due to smaller training process. Instead, we decided to divide the large dataset into 2 subsets - 5K and 10K, with images from the Visual Genome dataset.

Both the subsets of 5K and 10K images were trained with the same parameters and model structure. The images were resized to (256, 256, 3) and we were able to achieve favorable results compared to the larger 16.5K training. The 5K subset was trained for 300 epochs, and the 10K subset was trained for 102 epochs (due to Early Stopping). The below figure shows some success cases from both the 5K and 10K runs.

Figure 22: Success Case of Final Dataset, Top - Predictions, Bottom - Ground Truth

Our final dataset also consisted quite a few grayscale ground truth images. The model was able to successfully colorize these images with realistic predictions, as shown in the below figure.

Figure 23: Colorized Grayscale Original images. Top - Prediction, Bottom - Ground Truth

Additionally, the model also gave out quite a few failure output predictions for some categories of images. For objects in images that can take on multiple correct colors, the results ended up being brownish or sepia toned, which is mainly because of the L2 loss function. For example, in the below image, the umbrella takes on a multitude of colors. In order to minimize the overall loss of the model, a mean color value of brown is assigned to the objects.

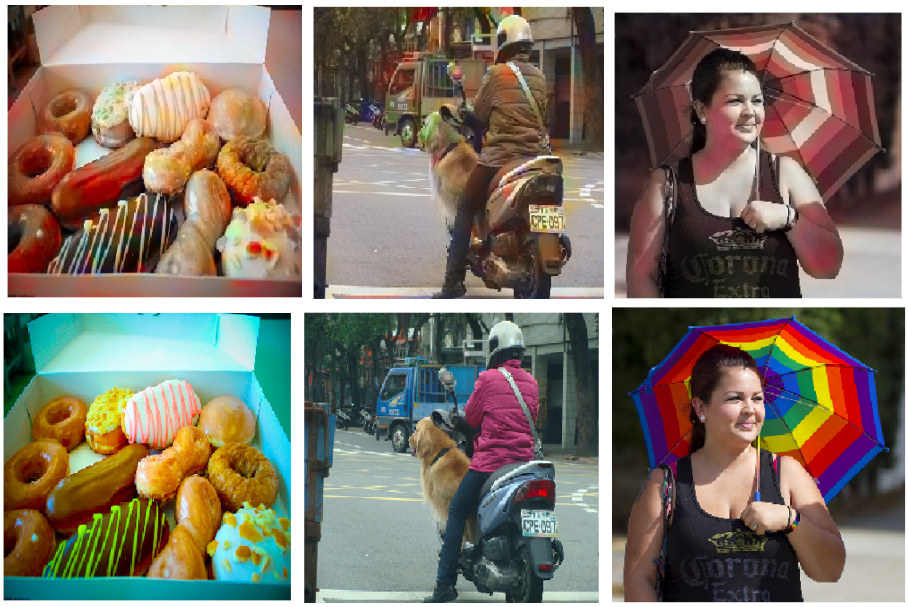

Figure 22. Final Data Failure Cases, Top - Predictions, Bottom - Ground Truth

Table 3. Performance of Inception-ResNet-v2 Transfer Learning Model

Dataset	L2 Loss			Mean Absolute Error			Epochs
	TRAIN	VAL	TEST	TRAIN	VAL	TEST	
Nature	0.0166	0.0859	0.0179	0.0908	0.1701	0.0938	100
Final 5K	0.0039	0.0132	0.0125	0.0391	0.077	0.0753	300
Final 10K	0.0081	0.01	0.0112	0.0595	0.066	0.0697	102

8 Conclusion and Future Work

In our paper we presented different deep learning model techniques to automatically colorize grayscale images. During our work, we design and build models step-by-step to make model more complex and generate better results. The Baseline CNN model helped us benchmark overall performance criteria, and improvements were made in subsequent Autoencoder and Transfer-Learning CNN models. It can be observed that the final Inception-ResNet-v2 model produced the best results overall. In fact, it produced realistic looking color predictions and was also able to colorize ground truth images that were grayscale with vibrant colors.

One of the key takeaways is the importance of datasets during training. With the Nature dataset, the domain was very specific and the final model was able to learn color information in just 100 epochs on 1K data. However, in our final diverse Visual Genome dataset, outdoor or natural images performed well, and indoor or multi-colored objects ended up with a sepia tone. Thus, the type of image data plays a significant role in training time as well as prediction quality. In general,

with enough data and trial times, the model would have been able to extract signature features and map to the right color.

For future considerations, a classification rather than regression approach would be useful and could help much more realistic color predictions, especially for some of our failure cases. This approach could also allow us to experiment with different loss functions rather than the L1/L2 functions used in our project. Additionally, implement GAN models for colorization could help make better diverse predictions and also make the image look sharper with better quality.

References

[1] McGill Calibrated Colour Image Database. http://tabby.vision.mcgill.ca/

[2] Visual Genome API. https://visualgenome.org/api/v0/api_home.html

[3] 17 Category Flower Dataset - University of Oxford. http://www.robots.ox.ac.uk/~vgg/data/flowers/17/

[4] Wallner, E. (2017, October 12). How to colorize black & white photos with just 100 lines of neural network code. Retrieved from https://www.freecodecamp.org/news/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d/

[5] Zhang, Richard, Phillip, Efros, & A., A. (2016, October 5). Colorful Image Colorization. Retrieved from https://arxiv.org/abs/1603.08511

[6] Iizuka, S., Simo-Serra, E., & Ishikawa, H. (2016, July 4). Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. Retrieved from https://dl.acm.org/citation.cfm?id=2925974

[7] (2018, November 13). Colorizing and Restoring Old Images with Deep Learning. Retrieved from https://blog.floydhub.com/colorizing-and-restoring-old-images-with-deep-learning/

[8] Baldassarre, Federico, González, D., Rodés-Guirao, & Lucas. (2017, December 9). Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2. Retrieved from https://arxiv.org/abs/1712.03400