Image Colorization

Nikhil Pathak
UCSD ECE Department

Abstract

In this project, I take on the task of Image Colorization, primarily inspired by the paper "Image Colorization with Deep Convolutional Neural Networks"[1]. The goal of the project is to train a deep CNN that can take a gray-scale image and produce a colorized version of that image that is believable to a human eye. Gray scale images are effectively obtained through use of the CIELUV colorspace, where the L-channel (analogous to an R or G or B channel in the RGB colorspace) represents the "lightness" of the image which can be directly correlated as gray-scale. In the project, I train a deep CNN that takes advantage of skip-connections and transpose convolutional layers in order to reproduce U and V channels at input resolution. Combined, the input L channel and the predicted UV channels will form a colorized image. To train and evaluate the model performance, I used the Huber loss function (inspired by [1]). In this report, I cover the motivations and difficulties of the Image Colorization problem. I also speak on the papers that inspired my work, including some of the models they explored in addition to what I did. Finally, I discuss the different experimental CNNs that I use to colorize images, including the loss results and sample images of the colorizations that demonstrate each model's performance.

1 Introduction

The task I take on in this project is Image Colorization, which is the task of converting a gray-scale image (also known as black and white) to an image with color. There is a lot of motivation for solving such a task, including colorizing old photos or films from time periods where color in photography was not yet invented. Bringing color to previously black and white images is a task that is frequently done manually by some people through image editing, so use of a CNN like I develop in this project could bypass some of that manual work. Improving the quality and experience of viewing images is a relevant task with the rise of deep CNNs, and Image Colorization is one of the tasks I tackle via such a neural network model.

One of the important parts of this problem is understanding that I will be using the CIELUV color space instead of RGB. Effectively, the L channel of LUV can be used as a gray-scale representation (a lightness scale) of the colorized image, and so this channel will serve as the input to my model. The model is then simply a function to predict the U and V channels of the image (2 channel output) that can then be concatenated with the L channel input to form the colorized predicted image (all three channels of L, U, and V form the colorized image similar to how the R, G, and B channels of the RGB colorspace form a colorized image).

Another important part of the problem is understanding that the evaluation of the model performance is tougher than some traditional computer vision tasks. With the Image Colorization task, there exists multiple possible "correct" predictions for a colorization of an image. For example, if there is a grayscale image of an apple on a table, the colorized apple could take the color green or red and still be a "correct" prediction of the colorization. So while we still may use an L1 or L2 loss of the residuals between the predicted U and V channels against the real U and V channels, we must acknowledge that such a loss function can fall short in this task for cases like the green or red apple and lead to some interesting-characteristic predictions with low saturation (more explained in the sections below). Another tough aspect of the evaluation is that assigning a metric to the model predictions is not as straightforward as using Accuracy or IoU seen in other tasks. With multiple possible colorizations and the high-resolution nature of the output prediction, the metric commonly used in such a task requires human intervention (such as the “colorization Turing test” described in [2] which measures the percentage of human trials which failed to properly differentiate between the predictions vs. ground truth images)! Due to the lack of access to such trials for my project, I perform manual inspections to determine quality of predictions. This is a topic I revisit later in the paper.

2 Related Work

The main paper that inspired my project (titled “Image Colorization with Deep Convolutional Neural Networks”[1]) approached this Image Colorization task in two ways. First, they introduced a Regression Network that used a simple combination of L2 and L1 loss (called Huber loss) to drive their training. This regression network produced some good predictions for the authors, but as shown in the examples of their paper, the lack of saturation in the predictions was apparent. The reason for this dullness of predictions is because objects or scenes that could take on multiple colors and still look believable (an apple could be green or red) generally result in the regression-based model learning to output the mean of the possible colors (the predicted color of an apple would be in between green and red). These mean colors may lead to the lowest regression loss, but they result in colors with low saturation and dull viewing pleasure. However, as shown in the paper, the regression network is still quite complicated, and produces effectively colorized images (even more consistent than the patchy colorness seen in the alternate classification network predictions). As a result of the dull predictions made by the Regression Network, the team explored a classification network, where each pixel in the U and V space is predicted to be in a certain class/bucket, where there are 50 total classes (each class representing a bucket range of possible raw pixel values that the U or V channel can take on). So with 224x224 pixels, with each pixel containing softmax scores for 50 classes, calculating the sum of cross entropy losses (as the training objective) and gradients (for weight updates) at each pixel will be **heavy** computations in training. While this bucketing/classification approach to the problem does result in more saturated images with more vibrant colors due to the class based approach that does not allow the prediction to arrive at a mean color of possibilities, the examples shown in this paper show a lot of patchiness and lack of consistency in the predicted image that is inferior to the predictions of the regression network (in my opinion).

Another source that I referenced in my project was Ryan Dahl’s published website[3] that actually inspired the previous paper’s Regression Network. Dahl explores a lot of different architectures, but ultimately finds that the VGG16 encoding layers in addition to some upsampling layers (nearly the exact model I implement, so it will be discussed in more detail below) produce the best predictions. However, he notes again that the dull, or “sepia-toned” (as he coins it), natured predictions are a downside of a simple L2/L1 loss. In his sample prediction images, it is clear that the model is quite strong with grass, which is an interesting discovery! One key take-away from Dahl’s work (that I will later reference) was that he used ReLU activation functions throughout his network, with exception to his last convolutional layer with which he used a Sigmoid activation function to push the predictions in the range of 0 and 1.

3 Method

As explained in the previous section, there are two CNN models discussed in the paper[1] for this Image Colorization task – a baseline regression network and a classification network. In this project, I implement the baseline regression network.

The reason for choosing this regression network instead of the classification network is explained in much more detail in Section 2, but to summarize, the Regression Network (in my opinion) produced smoother and less choppy predictions at the expense of having some dullness to them. I personally find that smoother but duller images are more believable than choppy patches of more saturation. With this realization, alongside the heavy compute power required to train the classification network, I implement the regression network mentioned in the paper due to its strength of more consistent/smoother (less patchy) predictions from a model that is more feasibly trained given limited training resources.

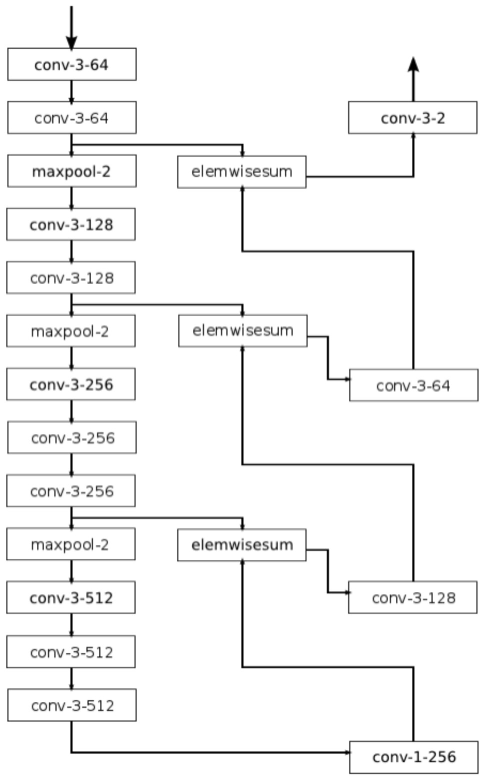

Figure 1: Regression Network Architecture (taken from [1])

With regards to the details of the regression network method I use, the architecture of the model is shown in Figure 1. To recap it, there exists a portion of the network that are pre-trained layers from VGG16 that is trained on ImageNet. This portion of the network is strong at featurizing an image for linear classification layers, so it can be reasoned that it will be effective for this Image Colorization task. This architecture is a similar approach to some Semantic Segmentation models, where we use an encoding architecture from VGG16 to represent an image at lower dimensions, but through use of upsampling (discussed in detail in the next section) and skip connections ("elemwisesum" in Figure 1), we bring the output prediction back to the original input image resolution. Throughout the model encoding layers, the use of max pooling effectively increases the receptive field of neurons at deeper layers while also reducing the number of required parameters in the model; it also allows the images to be encoding into lower dimensional features. The architecture uses ReLU activation functions (for non-linearity) and batch normalization in each "conv" block due to empirical performance. For the last forward convolutional layer of the network, I use the Sigmoid activation function (inspired by [3]) to ensure the predictions are within the 0 to 1 range for the UV channel predictions. With a deeper CNN, the use of ReLU and skip connections helps tackle the problem of vanishing/exploding gradients. With regards to the loss function that drives the training of the network weights and biases, the Regression Network (as done in [1]) uses the Huber loss function (also known as SmoothL1Loss in Pytorch[1]) which is a piecewise function that combines L2 and L1 penalty functions to tackle outliers more effectively ("less sensitive to outliers" as explained by Pytorch documentation). This Huber loss function has a threshold value for the residuals between the predicted and actual U and V channels: if the threshold is exceeded, use L1 penalty, else use L2 penalty. This loss function then dictates how the deep CNN weights and biases are trained through backpropagation (in order to minimize the loss)! It is important to note that I also use the Adam Optimizer for the training, as it is known for easy tuning and faster convergence[2].

[1]`https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html`

[2]UCSD ECE 176, Lecture 6, Page 19

Since we use VGG16 layers at the shallow end of the network (which is trained with 3 input channels per image), the model then takes in 224x224x3 input, so I will have to pass in 3 repeated gray scale representations of the image (as done in [1]). VGG16 is also trained on normalized input, so to accomplish this, I found the mean and standard deviation of the images in the dataset after they are converted to gray-scale. I can then use these statistics to normalize the gray-scale input (although the model was trained on normalized RGB input, this normalization step is still necessary to reduce the magnitude of raw pixel values and introduce negative inputs for more effective training that is done with additional finetuning on the pretrained RGB-input VGG16 layers).

As explained in section 1, the final evaluation of the model for the Image Colorization task is tough to quantify. Due to my inability to conduct any sort of "colorization Turing test"[2] (which requires large human trials) to obtain a solid metric, I evaluate and compare my model performances by use of the Huber loss. The assumption here is that the loss would be an accurate indication of model performance (lower loss is a stronger model), but this may not necessarily be true in terms of believability of the colorization (dull predictions might not be believable). But the loss is the best metric I have to work with given my available resources, and we are comfortable with predicting some dull images! I additionally perform manual inspections of the model predictions to subjectively determine which model is stronger. For the most part as discussed in detail in later sections, I find that lower loss leads to more believable predictions.

4 Experiments

4.1 Dataset

The dataset I use to train, validate, and test my model is the Flickr8k dataset[4][5]. I obtained the dataset from Kaggle[5], but it was originally created by a different team[4].

The Flickr8k dataset contains around 8000 images from different scenes and environments, as its main purpose (at least on Kaggle) is Image Captioning. However, for our task, I repurpose the dataset for Image Colorization since any colored-image dataset could be used for my task. In this case, Flickr8k was easily accessible through Kaggle while also not being too large or small in size that would cause poor model performance or long training times respectively. This serves as a really good dataset for the training because the environments and scenes used to train and test image-captioning models are usually the type that we might want to colorize from history, with either people or animals as the focal point for the captioning. The data format of the images of the Flickr8k dataset is JPEG. The size of the images are not constant, with each image potentially being square or rectangle with their own dimensions. One constant in the image sizes is that the width and height will be less than or equal to 500 pixels (there is no image with a width or height that exceeds 500 pixels). For our use case, we can simply resize these images to be the 224x224 input required for the VGG16 layers, potentially sacrificing the original aspect ratio of images for the purpose of focusing on the image colorization aspect of the project. Each image in the dataset also comes with a set of captions (for image captioning models), but I will not be needing those as the U and V channels extracted from the LUV input image (simple conversion from RGB to LUV) will serve as the targets for the gray-scale input. For training and evaluating my models, I split the Flickr8k dataset into training, validation, and testing splits of proportion 75%/15%/10% respectively.

As mentioned in Section 3, another important detail of the data used to train the model is that VGG16 requires an input of 3 channels for a traditional RGB image (VGG16 originally trained for image classification). Considering that we are only using the L channel (gray-scale) as our input, I simply repeat the L channel 3 times to form the model input in the same manner as the paper I am referencing[1]. Additionally, VGG-16 expects the input data to be normalized, so I found the mean and standard deviation of the images in the dataset after they are converted to gray-scale in order to normalize the input. For reference, the following were the statistics used to normalize the gray scale input (which lies in the range of 0 to 1):

$$\text{mean} = 0.4697$$
$$\text{standard deviation} = 0.2610$$

4.2 Best Model and Results

As mentioned before, the model I train takes in 3 repeated channels of normalized gray-scale image input (the L channel of the LUV image representation) at 224x224 pixels in order to predict the 2 channel output for U and V channels. Then we can concatenate the original single unnormalized L-channel with these predicted UV channels to form our colorized image prediction for viewing pleasure. The Huber loss function strictly acts on the ground truth U and V channel targets measured against the predicted U and V channels from the model. The loss is averaged over every pixel, so its magnitude is quite small.

One important thing to note is that in Figure 1, the original authors of the paper used Upsampling[3] layers in order to bring the encoded image back up to input resolution for the output. However, in my experimentation (which I will discuss more in the next section), I find that the model performed better (lower loss and higher visual quality of predictions) when I use transpose convolutions[4] to bring the image back up to high resolution.

In order to tune each of the models/training strategies I attempted to use, I formed a validation set from Flickr8k that acts as a form of unseen data. I employ early saving techniques in which I save the model with the best validation set performance (lowest loss) in the case that the model begins to overfit in later epochs. This early saving technique is common to ensure that the final resulting model is not one that is overfit on training data, and maintains the best generalizability on the unseen validation set. I also have a test set that I use to compare fundamentally different models' performances. This serves as a final metric for the strength of a model, and tells me which model is the best.

At the end of this all, I found the best model was trained with the full dataset using Transpose Convolutional layers (instead of Upsampling layers) and decaying learning rates. Specifically, I employed the Adam Optimizer with a learning rate scheduler[5] (ReduceLROnPlateau) that decays the learning rates when the validation loss increases for a patience epoch of 1. Through the hyperparameter tuning process via the validation set, I found that using different learning rates for different parameters of the network produced the lowest validation loss. In particular through tuning, the pre-trained VGG16 layers had a very small learning rate of 3e-6 for small fine-tuning on our task for the already strong feature extracting layers. Then, the remaining transpose and forward convolutional layers I added had a higher tuned learning rate of 1e-3. Finally, I use **no** regularization to fight overfitting, because introducing weight decay reduced the performance of the model on unseen data. The Figure 2 shows the training and validation loss curves for the 25 epochs that lead to a plateau in the learning for the best model that achieved the least test loss of 0.001849. It is important to note that while the validation curves usually are always higher than the training curves, the first epoch or two show the training loss higher which can be explained by the low batch size (due to memory constraints) affecting the running totals of the batch normalization in early epochs or the training and validation sets differing in the difficulty of image colorization.

As I have mentioned previously in the report, this Image Colorization task is tough to quantify for performance, so in Figure 3, I show some visual examples of the model inputs, target, and ground truths that we can analyze.

In the predictions, we see some very visually appealing colorizations. Certainly, to a human eye you might be able to closely inspect and find small issues with the colorization predictions, but at first and cursory glances, the image predictions are quite strong. Specifically, we see from the first and last row, the green grass/trees, blue sky, and animals are colored in a very believable manner. For the first dog row, there is no way for the model to perfectly recall the shade of the dog's fur, but the model produces a very believable color for it. Generally from the previous papers ([1], [3]) that implemented a similar model as mine, they also found the model predictions to be quite strong with green grass and light blue skies–a similar phenomenon observed with my model predictions. Additionally similar to previous Regression models, my model tends to predict duller colors. We see this in the grass of the first row of images, but more so in the second row of images. The carousel holding the kids is predicted to be quite dull in color (some shades of blue here and there are noticeable at closer investigations of the image) compared to the ground truth image. Such dullness is simply an effect of

[3]`https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html`

[4]`https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html`

[5]`https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate`

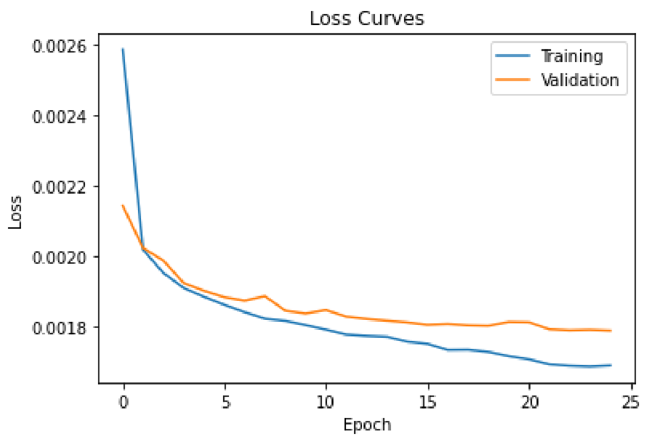

Figure 2: Loss Curves for Best Model - **Test Loss = 0.001849**

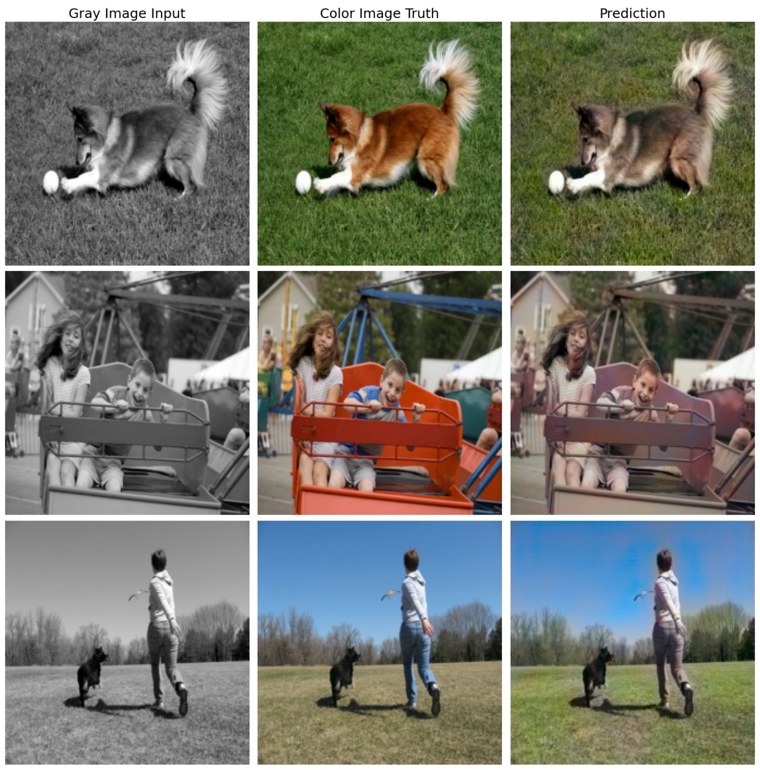

Figure 3: Prediction Examples of the Best Model

the way the model learns. Considering we use a simple combination of L2/L1 loss, the model sees a carousel that could be any color from blue to red to green (similar to how an apple could be green or red). To minimize the loss, it predicts an average of these colors which rounds out to be dull in nature. We see the dullness show up again in the jeans of the man standing next to the dog in the last row. So while the model is strong in some areas in producing believable colorized images, it also is weak in predicting higher saturation images that are more visually appealing. However, it is worth noting that amongst all the models to be discussed in the next section, this best model also produced the most (or same level of) appealing and believable predictions from my visual inspection (as mentioned prior, some manual supervision is required for the Image Colorization model evaluation).

4.3 Ablation Study

In the process of searching for the model with best test set performance (and tuning parameters through the validation set), I attempted multiple experiments seen in the table below and in Figure 4 to analyze which components or training strategies help or hurt the model performance. For Figure 4, I chose an input image that really highlights the different experiments' predictions! Visually, I would say that the (rough) order of lowest test losses is also the order for highest quality/most believable predictions on this input image (lowest loss = best prediction!). Below, I will briefly detail what each experiment tested. It is worth noting that the best model's prediction on this image is not great, but in my subjective view, it is better than the rest! Clearly, fully **dark blue** skies give the model some trouble.

Experiment	**Test Loss**
(1) ReLU Last Layer	0.003386
(2) Upsampling Layers	0.001934
(3) Half Training Set	0.001930
(4) Regularization	0.001890
(5) Universal Learning Rate	0.001876
(6) Best Model (from before)	0.001849

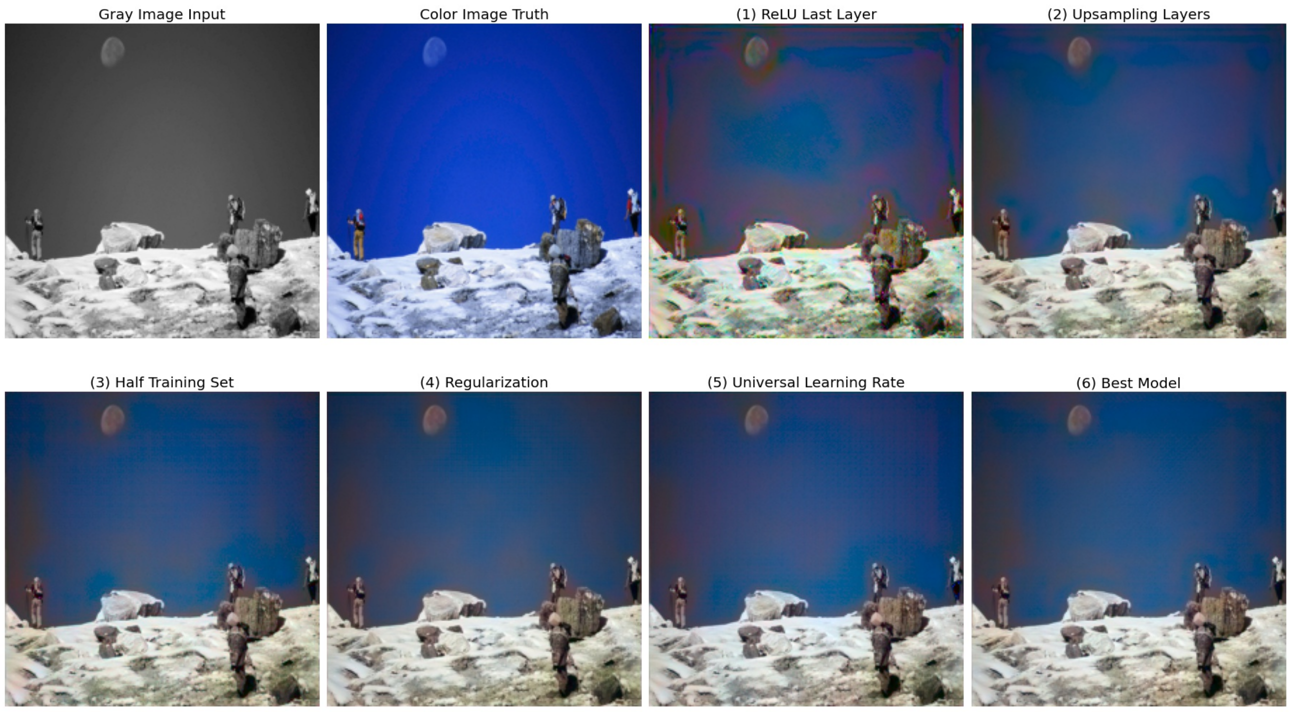

Figure 4: Experiments Visualized

4.3.1 ReLU Last Layer

In this experiment, I investigated using a ReLU activation function in the last convolutional layer instead of a Sigmoid function. This was one of the first models I tested, and it shows some of the worst results both in the test loss and the visualized prediction. In the Figure 4 prediction for this experiment, you can see a lot of color patches in the sky and a weird inconsistency/blurriness in the image that makes the prediction stand out as the clear worst amongst all the experiments. Clearly, this activation function is not the optimal one for the last layer due to its empirical performance seen through test loss, but also logically because the values for the U and V channels should be in the range of 0 to 1, but using ReLU only enforces the 0 bound–not the upper bound of 1. Using a sigmoid function is clearly superior for this Image Colorization task.

4.3.2 Upsampling Layers

In the inspiration paper[1], the authors used upsampling layers to bring the prediction back to the 224x224 input size. Logically, this makes sense because we want to ensure the output is at high resolution, but from my experimentation, I found that using transpose convolutional layers were more effective. With upsampling layers, we simply scale the image up using a bilinear interpolation algorithm, so there is **no** learning in this operation, whereas transpose convolutions have parameters that can be optimized for the given task. Due to this, we see the test loss for the model with upsampling layers is clearly higher than models with transpose convolutions used. The actual image prediction in Figure 4 is not significantly worse in quality than some other experiments, but compared to the best model which used transpose convolutions, it has a lot more color patches in the sky that make it a less believable colorization of the gray image.

4.3.3 Half Training Set

With this experiment, I simply tested the effects of varying the size of the training image set. Specifically, I halved the dataset used for training, and it produced a test loss that was much higher than the best model. This is something that we can intuitively understand because more data in the training set would force the model to be more generalizable to unseen data and also give the model more information to learn from. Presenting more types of images or scenarios lets the model learn to be more equipped for such images in the unseen datasets, but it also gives more confidence that the training set would have less percentage of outliers. While it may increase training time, adding more data clearly improves the test loss. Moreover, if you inspect the corresponding experiment prediction in Figure 4, you can see some more noticeable bad color patches, such as next to the rightmost person in the image. While the quality of the image is not much worse than the best model, the test loss is clearly lower, and there is some small noticeable quality difference in the predictions.

4.3.4 Regularization

This experiment was more so tuning hyperparameters, but I included it here anyways because I felt it was an important revelation. The only difference between this model and the best model discussed in Section 4.2 is that this model was trained with regularization included in the optimization process. Regularization is a technique to fight overfitting to the training set, so with a higher regularization strength constant, we would see the training and validation loss curves stick more closely together over the epochs because the model would be penalized for learning complex weights that are meant to only fit to the training set (and therefore lose generalizability to unseen data like the validation set). However, it is clear from the test loss results that the regularization actually hinders the learning of the model that could be applied to unseen data. In some cases, learning complex weights can be generalizable to unseen data in addition to the training set, and this seems to be one of those cases where useful model complexity is being punished and therefore model performance suffers. While the test loss is higher for this model compared to the best model, we also see the corresponding prediction in Figure 4 has more area of mis-colored patches. For example, the space over the highest rock in the image and the right edge of the image have clearly more area of non-blue patches that make the model less believable as a colorization compared to the best model.

4.3.5 Universal Learning Rate

This experiment was also just tuning hyper-parameters, but led to some interesting discoveries worth discussing. In this test, I tuned a universal learning rate for all the parameters in the model, including the ones from the pre-trained VGG16 layers. This contrasts the best model, which had smaller learning rates for the pre-trained layers compared to the layers I added on top. The resulting test loss from this trained model was higher than the best model, which tells us that taking larger steps for pre-trained parameters may not be worth it! The early layers of VGG16 are trained on the huge ImageNet dataset which means it is a powerful and generalizable feature extractor. Making large weight updates on these pre-trained parameters could simply lead to overfitting to the training set. Small fine-tunings of these parameters to adjust for the Image Colorization task and the gray-scale input (instead of RGB) are ideal based on my tuning. Clearly, taking larger steps negatively affects the model performance likely because the VGG16 feature extractors are being modified so much to the point of losing generalizability obtained from ImageNet training. Visually, we see the prediction in Figure 4 has even more area of non-blue color patches in the sky compared to the previous 2 experiments. In my opinion, this model's prediction is a downgrade in quality and believability compared to the best model.

References

[1] Jeff Hwang and You Zhou. Image colorization with deep convolutional neural networks, 2016. `http://cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf`.

[2] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization, 2016.

[3] Ryan Dahl. Automatic colorization, 2016. `https://tinyclouds.org/colorize/`.

[4] Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. Collecting image annotations using Amazon's Mechanical Turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 139–147, Los Angeles, June 2010. Association for Computational Linguistics. Original Producers of Dataset Posted by Aditya Jain.

[5] Aditya Jain. Flickr 8k dataset, 2020. `https://www.kaggle.com/adityajn105/flickr8k`.