CS577 Spring 2020 - Final Project

# Global and Local Image Priors for Automatic Image Colorization

Niti Wattanasirichaigoon
A20406934

## 1. Introduction

Research on automatic colorization of grayscale images has extensively explored in the fields of computer vision and machine learning. As of present, many researchers were able to produce impressively accurate colorization models using convoluted neural networks (CNN). The ideal colorization network merely intakes a black and white image, and outputs an accurate colored version of it.

Among the research on automatic colorization, a model proposed by Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa in their paper '*Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification*' (2016) [1], was able to produce stunningly accurate colorizations compared to the state of the art. The researcher's proposed model used slightly different practices than traditional colorization models, including an implementation of an image classifier to serve as an additional dimension of data for the colorization model.

In this project, I implemented the model proposed by the aforementioned paper in Keras and tested if I could get similar results. I then include a discussion on how my setup and results differ from the researcher's, how can I improve it, and instructions on how to run my model in Python.

## 2. Related Works

Traditional image colorization models requires additional input from the user such as color scribbles to serve as a guideline [2]. The algorithm then continues to colorize an image within a closed area until an edge is detected by solving a quadratic cost function between neighboring pixels. This method was later improved in succeeding research that prevents color from bleeding across boundaries [3]. The major downside of this method is that we need some kind of color guidance from the user.

On the contrary, fully automated methods focus on training models over similar images [4]. These so called 'color transfer' techniques allows to model to accurately colorize images based on its trained color scheme. These models may be able to create accurate colorization results, but only

for images that are similar to the data it used for training. Such models would have to be extensively trained to a large variety of images in order to accurately colorize different objects. More recently, Charpiat [5] proposed the use of a global feature to help predict the possible colors of each pixel. This improvement allowed colorization models to distinguish and more accurately colorize images of different global features. The downside still lies that it requires some kind of global feature input from the user.

## 3. Proposed Solution

The researchers of our main paper proposed a solution where the model also incorporates global image priors for image colorization [1]. The model does not require any additional input from the user other than the black and white image. Additionally, it can process images of any resolution as opposed to many previous research that can only process images of a fixed resolution. The model can be described as an image colorizer that implements an image classifier inside it to extract global image priors. The model was trained over a large dataset of various images with 'labels' in an end-to-end fashion. During inference, the model will predict the label of the input image in its classification network and use it as a prior for colorization by fusing it with the local features network.

### The Model

The architecture of the model in our paper can be seen below. It is mainly divided into four components, the Low-Level Features Network, the Global Features Network, the Mid-Level Features Network, and the Colorization Network.

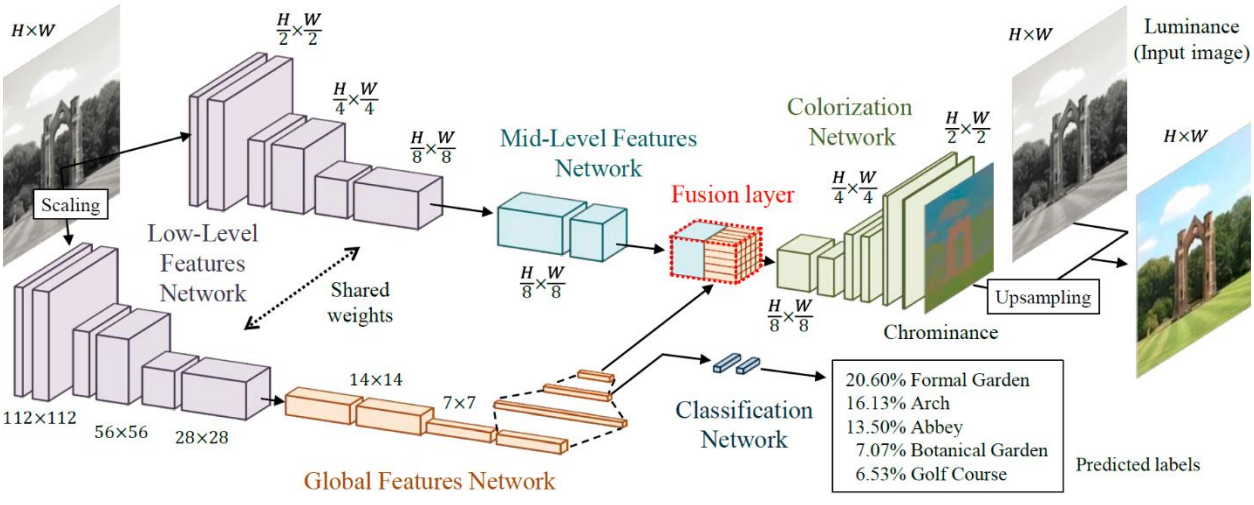



**Figure 1:** Overview of proposed automatic colorization model.

Further description of the model architecture is represented in the following table.

(a) Low-Level Features network

| Type | Kernel | Stride | Outputs |
|---|---|---|---|
| conv. | $3 \times 3$ | $2 \times 2$ | 64 |
| conv. | $3 \times 3$ | $1 \times 1$ | 128 |
| conv. | $3 \times 3$ | $2 \times 2$ | 128 |
| conv. | $3 \times 3$ | $1 \times 1$ | 256 |
| conv. | $3 \times 3$ | $2 \times 2$ | 256 |
| conv. | $3 \times 3$ | $1 \times 1$ | 512 |

(b) Global Features network

| Type | Kernel | Stride | Outputs |
|---|---|---|---|
| conv. | $3 \times 3$ | $2 \times 2$ | 512 |
| conv. | $3 \times 3$ | $1 \times 1$ | 512 |
| conv. | $3 \times 3$ | $2 \times 2$ | 512 |
| conv. | $3 \times 3$ | $1 \times 1$ | 512 |
| FC | - | - | 1024 |
| FC | - | - | 512 |
| FC | - | - | 256 |

(c) Mid-Level features network

| Type | Kernel | Stride | Outputs |
|---|---|---|---|
| conv. | $3 \times 3$ | $1 \times 1$ | 512 |
| conv. | $3 \times 3$ | $1 \times 1$ | 256 |

(d) Colorization network

| Type | Kernel | Stride | Outputs |
|---|---|---|---|
| fusion | - | - | 256 |
| conv. | $3 \times 3$ | $1 \times 1$ | 128 |
| upsample | - | - | 128 |
| conv. | $3 \times 3$ | $1 \times 1$ | 64 |
| conv. | $3 \times 3$ | $1 \times 1$ | 64 |
| upsample | - | - | 64 |
| conv. | $3 \times 3$ | $1 \times 1$ | 32 |
| output | $3 \times 3$ | $1 \times 1$ | 2 |

First, a black and white input image enters the network where it gets split into an input for local image priors and an input for global image priors. The original input may be of any resolution, it will then be cropped to a size of 224 x 224 before it enters the global image prior route. The image input and outputs used are in the L*a*b* or (CIEL*a*b*) color space and not RGB. The L*a*b* color space also consists of three channels a luminescence channel (L*) containing floating values between 0 (black) to 100 (white), and two color channels (a* and b*) with values ranging from -128 to +127. The network receives a normalized luminescence layer as an input, and predicts the two colored channels in the end. To visualize the output, we have to combine it with the luminescence layer again, denormalize the channels, and then transform it back to RGB so that we can plot it with *Matplotlib*. The transformations from L*a*b* to RGB and from RGB to L*a*b* color spaces can be done via *lab2rgb()* and *rgb2lab()* methods in the *skimage.color* module. The researchers experimented with various color spaces (RGB, L*a*b*, and YUV) and found that the Lab color space gave them the most accurate results.

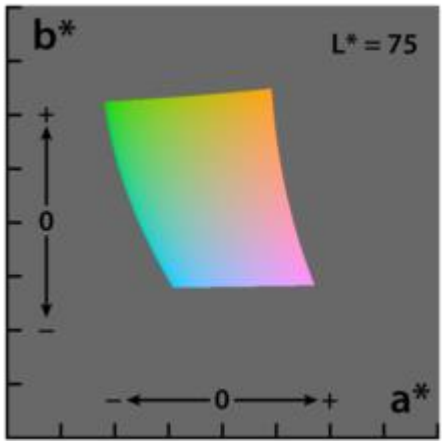



**Figure 2:** L*a*b* color space. Image from *Wikipedia*.

The low-level features network is composed of 6 convolution layers where the weights are shared between the local and global image prior routes. The image widths and heights are halved three times by convolutional layers of stride = 2 with 'same' padding instead of using max-pooling to maintain performance [6]. The mid-level features network is merely two more conv layers for the local image prior route.

The global features network includes some conv layers before flattening and connected to three dense (fully connected) layers. The output of the second dense layer splits out to our classification network where the label of our image will be predicted. This output utilizes a categorical cross-entropy loss.

The output from the last dense layer is then repeated and concatenated after every coordinate of the output of the mid-level features network in the fusion layer. A 1x1 conv layer is then applied to integrate the data from our two branches together.

The combined data tensor enters the colorization network where it passes more conv layers and up-sampling layers. The up-sampling layers allow the image tensor to return back to its original dimensions and delivered as an output. The output layer has two channels for the a* and b* components of the color space and the loss will be the MSE between the output and the ground truth a* and b* channels.

### Optimization and Learning Parameters

Each layer in the model is followed by a batch normalization layer and a 'relu' activation except for the outputs where the classifier and color outputs use 'softmax' and 'sigmoid' activations respectively. The optimizer of choice of the researchers is ADADELTA as they claim that the learning rate will adjust itself and so we have one less hyper parameter to worry about.

### The Data

The dataset used to train this model is the MIT Places dataset [7]. It is an image dataset containing 2,448,872 images from 205 class categories. The images are of 256 x 256 resolution. The images will be converted to grayscale to use as inputs for the model. The original colored images and their scene labels will be used for model training. The testing images are taken from the validation set.

## 4. Implementation Details

There are some differences in implementations between the researcher's and my experiment. An overview of implementation differences is summarized in the table below.

| | Researcher's experiment | My experiment |
|---|---|---|
| **GPU Framework** | PyTorch | Keras |
| **GPU** | NVIDIA Tesla K80 | NVIDIA GeForce GTX 850M* |
| **Training Images** | 2,327,958 | 850 |
| **Validation Images** | 19,546 | 150 |
| **Classes** | 205 | 10 |
| **Batch Size** | 128 | 8 |
| **Iterations** | 200,000 | 3,180 |

**Table 2:** ** The GeForce GTX 850M can be thought as roughly 6-8 times weaker than the Tesla K80.*

Although I was successful at building the researcher's model in Keras, the experimental results are much different because of the difference in machine speculations and vastness of training data. Unfortunately, my operating machine does not have enough space to carry training of the full dataset (~24 Gb) nor do I have as much time to train the data as the researchers did (they trained their model for about 3 weeks). I decided to choose 10 distinct image classes and work with a smaller sample of the dataset instead (100 images of each class, split uniformly into 850 training and 150 validation images). Because of a lesser GPU, the maximum batch size that I was able to train with without causing GPU Out of Memory (OOM) Errors is 8. We will see later how my small batch size actually affects the performance of the model.

Regardless of the results of the direct implementation, I also built the exact same network but without the global features network and fusion layers. I wish to compare the performance of the model with and without the global image prior, to see how much it affects the colorization performance. Finally, I created a separate function to test the model on unseen images.

## 5. Results and Discussion

A total of 4 models were built and trained, three being a direct implementation of the method according to the paper and the fourth being the implementation without the global features network.

**Model 1:** *No batch normalization, equal loss weights*

The initial model was a direct implement of the architecture according to the paper trained over 10 epochs with a batch size of 5. Here we can see a decrease of the training loss (MSE) suggesting that there is some form of overfitting. The predictions, however, returned practically the same black and white image as the input, suggesting that there is some failure in the colorization process. Looking closely, the total model loss is computed as the sum of the loss of our two outputs. This is a problem because the loss calculated from cross-entropy is a lot higher than the MSE of colorized output (about 1000 times larger). Thus, the decrease in training loss we see in the plot in figure 3 is essentially the training process of the classifier network only.

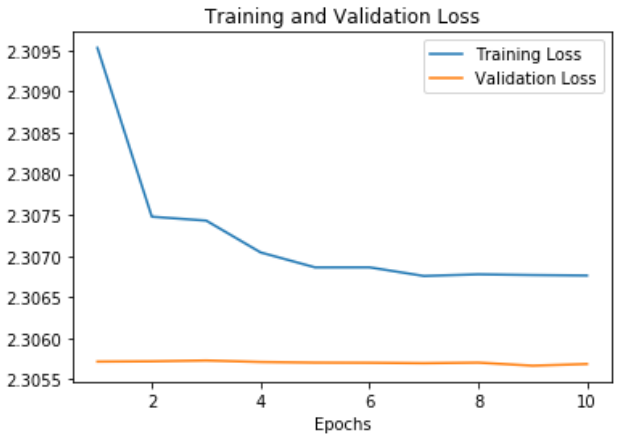



**Figure 3:** Training and validation loss of model 1

**Model 2:** *No batch normalization, adjusted loss weights*

The second implementation was train over 10 epochs with a batch size of 8. The loss weights of the outputs were adjusted so that total loss = 1000 * color output loss + classification loss. However, now we do not see any training in the model. The model returns practically the same image out as the output, suggesting that there is still no learning in the colorization network. We can only fic this process by adding a batch normalization layer after each layer before the transfer function to prevent the gradient from vanishing.

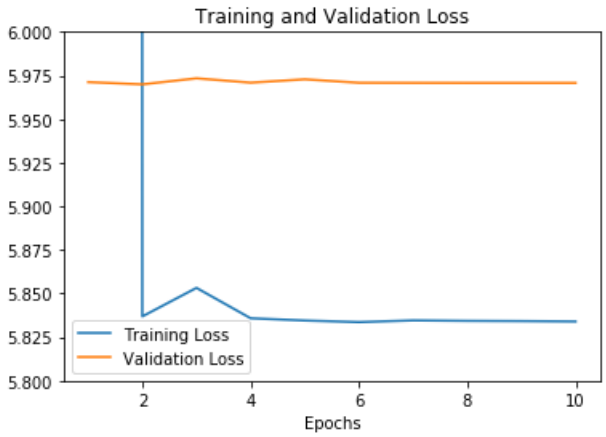


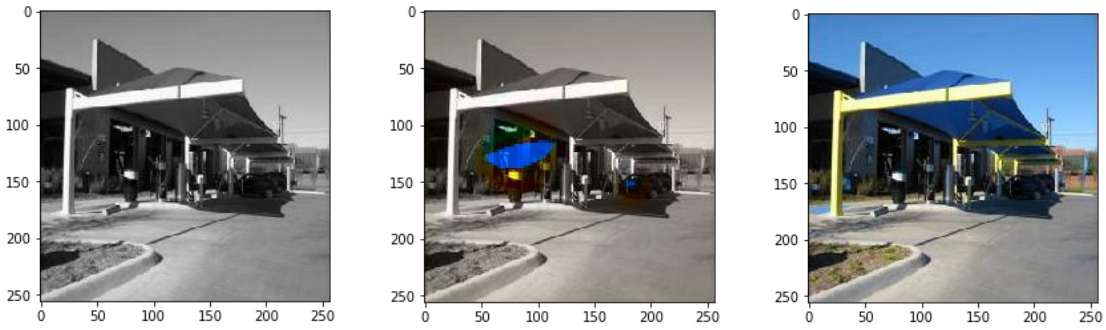

**Figure 4:** Training and validation loss of model 2. No learning observed.

**Model 3:** *Batch normalization and adjusted loss weights (Full Implementation)*

The third implementation includes batch normalization throughout the network and thus a gradient is present. The model was trained over 30 epochs at a batch size of 8. We can see a decrease in both the cross-entropy and MSE losses suggesting that there is learning of both the classifier and colorizer networks. The violent fluctuations in the validation loss is actually a good sign. It suggests that there are actually some right and wrong colorization predictions present. High variations in the validation loss is usually observed when our validation set is small or there is a high variance in it, which is the case because image data naturally has high variance.


**Figure 5:** Training and validation loss of model 3.

With a more promising model, I proceed to test the trained model on a random image from the validation set to observe its colorization performance.



Input Prediction Ground Truth

**Figure 6:** Model 3's predicted result of a random test image.

We can see in figure 6 that there is certainly some kind of activity occurring in the model, but it still fails to produce an appropriate colorization of the image. Most of the predicted images have a slight brownish tone given to the overall image and occasionally some color bleaches like the one seen in figure 6 – Prediction. Further training or changing of hyper parameters did not produce significantly better results from this model. We will come back to discuss the results of this model after model 4.

**Model 4:** *Full implementation without the global features network (classifier)*

A separate model 3 was built without the classifier branch. Since most of the model's parameters were contributed from the fully connected layers in the classifier branch (roughly 33 million out of 44 million), training of the model became much less computationally demanding for the GPU. I was able to train model 4 with a batch size of 16 for 20 epochs. Although we are missing the global image prior, the increase in batch size allowed the model to learn more differences between images, and produce more accurate colorizations. We can see that both the training and validation loss decreases with the number of epochs, suggesting that there is learning present.

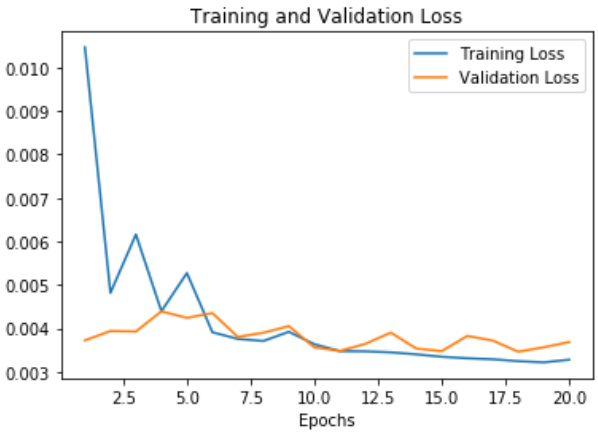



**Figure 7:** Training and validation loss of model 4.

Since batch size is the number of images that go through the network before an iteration of back propagation takes place, we can explain the difference between model 3 and model 4's colorization performances. Smaller batch sizes will allow for more frequent updates to the model and can cause the model to converge faster. Smaller batches also puts a less toll on the GPU

because less computations are held in memory at an instance of time (which is why my GPU can only run smaller batches while the researchers can run larger batches). The downside of small batch sizes is that the model would have to update its gradient with high variance batches. This can cause the model to fall into local minimums guided by the first few batches it sees and ultimately will fail make accurate predictions.

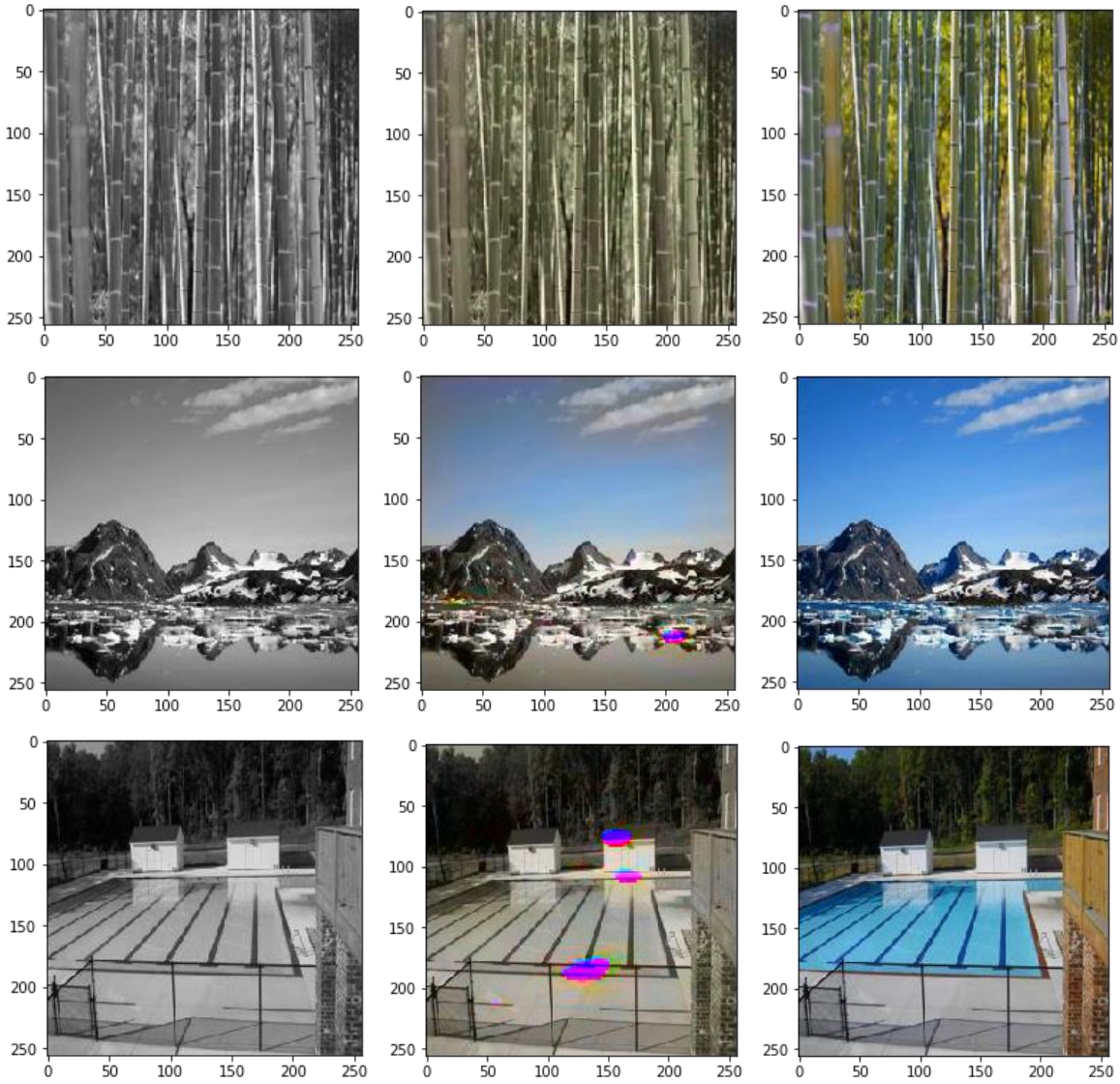



**Figure 8:** Model 4's predicted result of 3 distinct images including a bamboo forest, icy mountains, and a swimming pool.

Model 4 was able to produce much more satisfying results than model 3 although it is still far from perfect. We can see in figure 8 that when trained to clearly distinct types of images, it is able to produce the appropriate color in the colorization process. The model was able to color the bamboo forest with a green palette while also being able to color the blue sky in the second example. This represents a great improvement from model 3 as the model is able to distinguish between different features in images and assign different colors to them. The last example of the swimming pool shows that the model still produces failed results similar to those of model 3 when it is fed an image that is far from what it had seen during training. The model might have been trained with swimming pool images, but it might not have seen a pool image from this angle, causing it to fail to color the image properly.

**Limitations**

The main limitation of fully automated models like this one is that it highly depends on the training data. A vast and large data set for training is required for the model to make accurate predictions. Since my experiment used only 850 training images (85 from each class), it generalizes poorly compared to the researcher's model. As seen in the swimming pool example of figure 8, the model fails to colorize objects or structures that it has not seen before.

Another limitation is the ambiguity of color in grayscale images, especially with artificial colors. If different colors transform into the same shade of gray, then how can the model possibly know what is the true color of the image? We can see from figure 9 that even the researcher's model (taken from their web demo) fails to color the red flowers in the picture. To accurately colorize images like this, the model must be trained extensively on a large amount of similar training images. In this case, both my model and the researchers were able to detect the trees and apply some green colorization to the picture, but they could never know from the grayscale image, what color the flowers are.

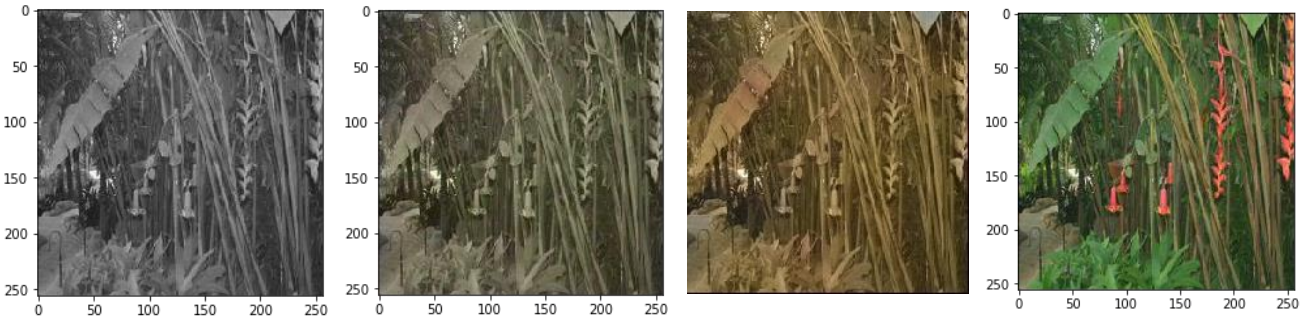



**Figure 9:** Predictions of an image with color ambiguity. Even the extensively trained model of the researchers failed to color the red flowers.

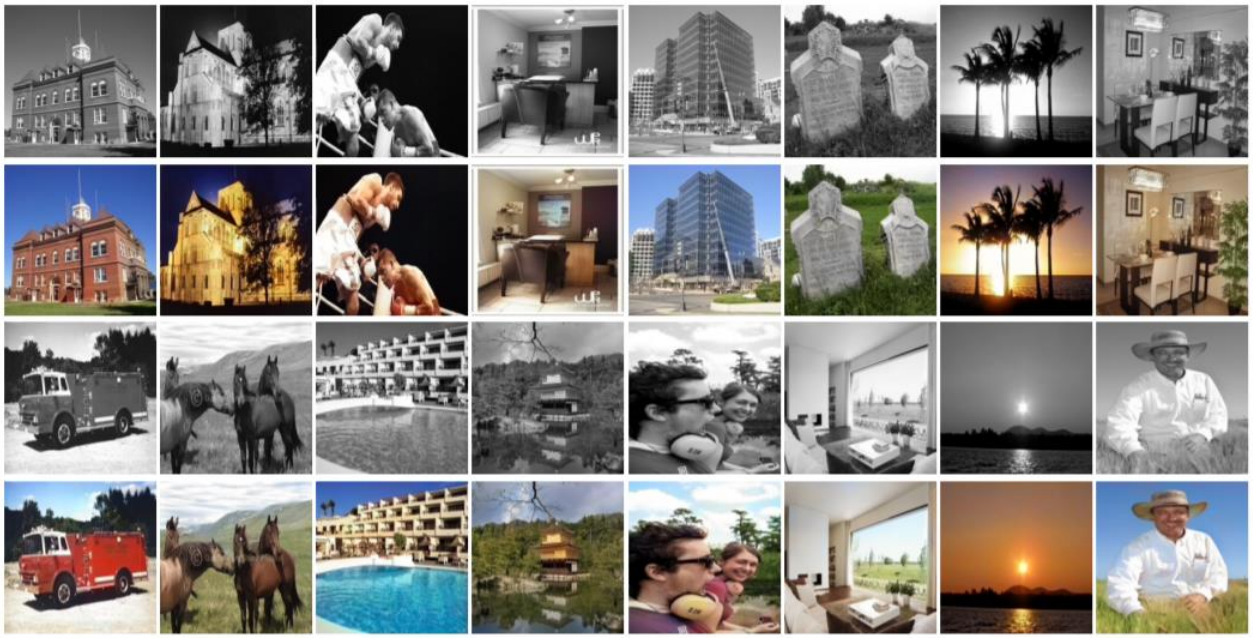


**Figure 10:** Sample predictions from the researcher's fully trained model. [1]

As an attempt, I tried to apply data augmentation to compensate for my small training set for training of the first three models but have failed. The model output must include the two color channels and the class label. The class label has dimensions different from that of an image and so Keras's *ImageDataGenerator* cannot be used. To implement data augmentation, I would have to code a custom generator myself.

## 6. Conclusion

I have implemented the researcher's CNN for automatic colorization but the model failed to produce accurate results possibly due to the small batch size and training dataset (model 3). We can see how in the implementation without the classifier was able to produce better colorization results just by an increase in batch size from 8 to 16 (model 4). We can hypothesize that if trained under similar GPU speculations and training data size, we would be able to achieve similar results like the researcher's (figure 10).

This study required me to utilize different neural network architectures learned from the course (functional API, conv, dense, Batch norm layers) as well as other techniques (up-sampling, fusion layers, multiple outputs), and I also learned how hardware limitations can affect the model training process. The lessons learned from this project will help guide me when I encounter problems involving CNNs that work with image data in the future.

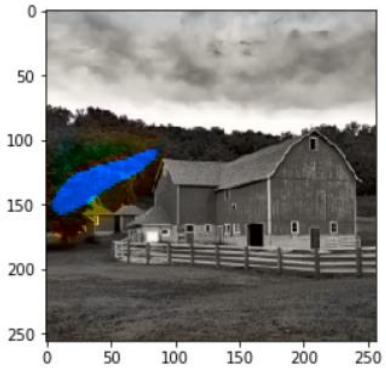
## 7. Program Demo Instructions

Below are the instructions to load the two trained models, the direct implementation model and the model without the classifier. In theory, the direct implementation should perform better if we have the correct tools for training. In my experiment, as described in the report, the model without the classifier performed better most likely because it was able to train with larger batches.

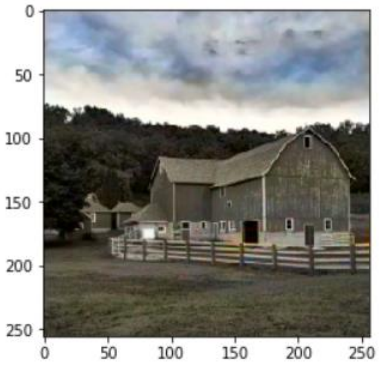
1. Download the zip file from https://drive.google.com/open?id=1f02NORnkbIUfhjPg6bmRY5Yj-lPglvcc. (size ~500 Mb, please use IIT account)
2. Unzip the folder, there will be a python notebook file, two models, and six black and white images for testing taken randomly from google.
3. Start the python notebook CS577_Project_NitiW_Demo.ipynb
4. Run all the cells in the notebook. Use the `predict_color` function in the 4th cell to test the two colorization models on different test images.

Example:

test_img5.jpg
(input)




colorizer1




colorizer2

To test the researcher's model demo, go to: http://iizuka.cs.tsukuba.ac.jp/projects/colorization/web/

## 8. References

[1] Satoshi Iizuka, Edgar Simo-Serra, Hiroshi Ishikawa. 2016. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. SIGGRAPH.

[2] LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. ACM Transactions on Graphics 23, 689– 694.

[3] HUANG, Y.-C., TUNG, Y.-S., CHEN, J.-C., WANG, S.-W., AND WU, J.-L. 2005. An adaptive edge detection based colorization algorithm and its applications. In ACM International Conference on Multimedia, 351–354.

[4] REINHARD, E., ASHIKHMIN, M., GOOCH, B., AND SHIRLEY, P. 2001. Color transfer between images. IEEE Computer Graphics and Applications 21, 5 (sep), 34–41.

[5] CHARPIAT, G., HOFMANN, M., AND SCHOLKOPF, B. 2008. Automatic image colorization via multimodal predictions. In ECCV

[6] SPRINGENBERG, J. T., DOSOVITSKIY, A., BROX, T., AND RIEDMILLER, M. A. 2015. Striving for simplicity: The all convolutional net. In ICLR Workshop Track.

[7] Places, the scene recognition database. MIT Computer Science and Artificial Intelligence Laboratory.