# IMAGE COLORIZATION USING CONVOLUTIONAL AUTOENCODER

*Zahid Hossain*

## ABSTRACT

The task of colorizing grayscale images takes tremendous skill and time. Professional colorists can take up to a month to colorize a photo. What if a machine learning model could do the task for us in mere seconds? This paper presents my exploration of training a neural network to perform the task of image colorization using PyTorch.

## 1. INTRODUCTION

There are endless ways to approach an ML colorization problem. In any case, one must prepare a dataset to train on, select or develop a model, fine tune parameters during training, and finally evaluate the results.

## 2. SOURCE DATASET PROCESSING

I chose to use my own personal images to create the dataset. It contains approximately 1500 images that comprise mostly of natural landscapes, but other scene types are mixed in. All the source images were first compressed using the open-source Caesium software to reduce their file size. This was done in an attempt to speed up any disk IO-bound operations. I then created a Python script to split, resize, and pad all the images. I split them at random such that 90% were used for training and 10% for validation. The images were resized to 2000x2000 squares with zero-padding. I took advantage of Python's multiprocessing library to parallelize this task, achieving nearly 15x speedup when compared to a single-threaded for-loop implementation (1min vs. 15min).

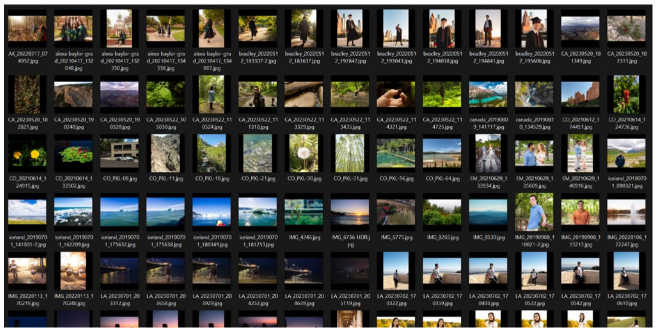

Fig 1. Screenshot of portion of dataset

## 3. MODEL SELECTION

When researching models for colorization, I primarily saw the use of either convolutional autoencoders or GANs. I chose to use an autoencoder model because it seemed easier to train, but the results from GAN based models tended to look more realistic.

### 3.1. Autoencoder Layers

An autoencoder can be described with an encoder-decoder structure. The encoder compresses the input into a smaller latent space representation. The decoder will then upscale the latent space back to the same size as the input. For my model, the encoder was implemented using the first 6 layers of the ResNet-18 model [1]. The decoder was then implemented using the same type of convolutional layers but in reverse, with upsampling. This model design was inspired by Luke Melas-Kyriazi's research [2]. His model used ResNet with 365 weights, but I changed it to use the default ResNet because I found it had very slightly improved color output.

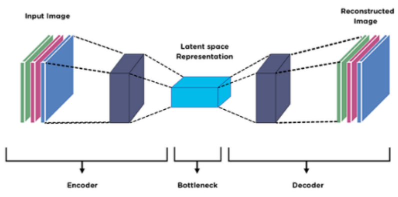

Fig 2. Autoencoder model, Source: [3]

### 3.2. Color Space Choice (LAB vs. RGB)

The model takes a 1-channel grayscale image as input and computes multiple channels to create a color image. Color images are traditionally stored in RGB format where each of the 3 channels store color information. We could theoretically have our model output 3 channels of information, but we can reduce the training complexity by using a different color space. The LAB color space stores color in 2 channels (A and B). The L channel stores luminance (grayscale) information. By choosing to train the model to output AB channels, you can recombine the predicted AB channels with the input grayscale image to reconstruct a full color image.

## 4. PYTORCH DATALOADER PREPARATION

Because I was using my own local dataset, I had to create a custom class inheriting PyTorch's Dataset class. I implemented a customized getitem() method that would return separate tensors for L and AB channels used for training. This class can then be passed to a PyTorch DataLoader.

## 5. TRAINING

I started training by using 224px (side length) images for speed (~1 min/epoch). I experimented with the learning rate and batch size until I got reasonable results. I then switched the input to 608px. With a batch size of 15 and learning rate of 0.0001, the training took about 4 minutes per epoch. Training was done on my laptop with an Nvidia RTX 3080 GPU.

## 6. RESULTS

Below are several figures with the results of the model. It doesn't perform very well with people and skin, and the images tend to be desaturated. You can at least see that skies are relatively blue, foliage is green, and ground is earthy. This is most likely influenced by the dataset's small size and landscape/nature bias.

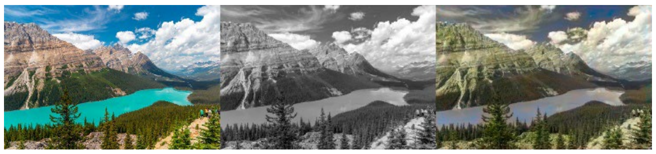

Fig 3. Left: original, Middle: grayscale input, Right: colorized output (from training validation)

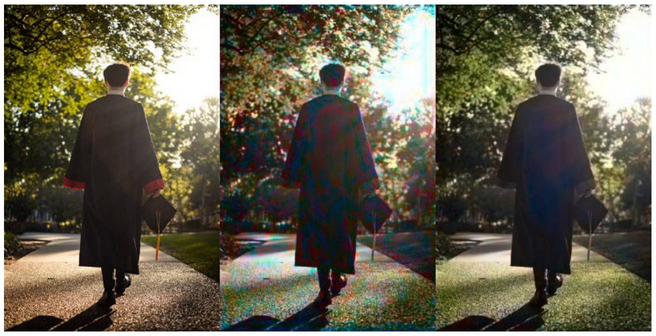

Fig 4. Left: original, Middle: model output after epoch 1, Right: output after epoch 16

## 7. REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition", 10 Dec. 2015, https://doi.org/10.48550/arXiv.1512.03385

[2] Melas-Kyriazi, Luke, "Image Colorization with Convolutional Neural Networks", 15 May 2018, https://lukemelas.github.io/image-colorization.html

[3] Birla, Deepak, "Basics of Autoencoders", 12 Mar. 2019, https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f

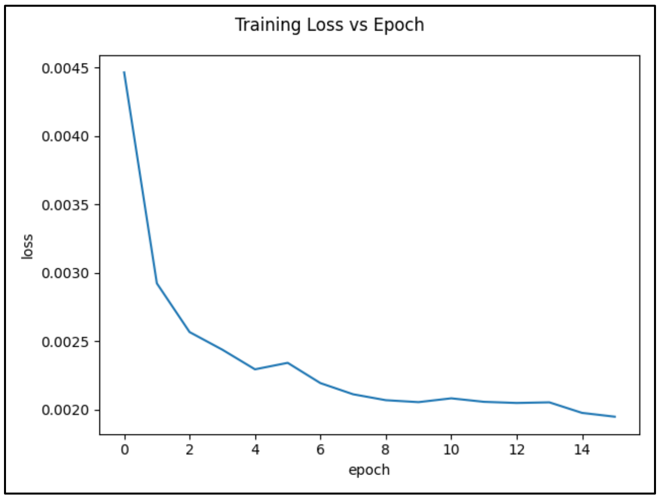

Fig 5. Training loss after 16 epochs