Image Colorization

Nakhul Kisan
Electrical and Computer Engineering
A15217974

Abstract

Using some grayscale image as input, we will attempt to prescribe it some possible colorization without significant user input and without desaturation of these colors using a convolutional neural network in our machine learning architecture. We will largely follow the architecture outlined by the Zhang et al. paper.

1 Introduction

The purpose of this learning algorithm is not to guess the original colors of an image back onto a grayscale version of the image. Rather, the algorithm will recreate a possible color scheme for an image that could be used to pass a Turing Test and fool a human observer. Our understanding of the model is that our convolutional neural network will try to guess the underlying features of the grayscale image and then map those features to some color values in an a*b space.

The model does not train in an RGB color space. Instead, our training is done in an L*a*b color space, popularly known as the CIELAB color space, where L is the lightness level of each pixel, and (a, b) both correspond to the 4 colors of human vision. This is done by taking some grayscale image as our input and then guessing some a and b values for that image. In our final prediction, we will convert this a*b prediction from our CNN[1] into an RGB image by concatenating the 3 channels (L, a, and b).

2 Method

The convolutional network model was built based off of the architecture described on page 4 of Zhang et al. by utilizing the PyTorch API for convolutional layers, batch norms, etc. and running through the entire architecture via inheritance from the nn.Module class.

Model Structure:

- I tried to follow the structure of the model outlined by Zhang et al. as closely as possible. I used a grayscale image as their input (channel size = 1), and then learn 64 features of the grayscale image after a pass on the first convolutional block. This means that our input channel dimension to conv1 is 1 and our output channel dimension is 64. The number of channels is continuously doubled until 512 is the dimension of our channel space. This was learned over the course of 4 convolutional blocks (using nn.Sequential) of 2-3 convolutional layers each.
- The reason the channels are expanded to reach 512 is because, for L*a*b image representation, a is within the range [-128,128] and b is within the range [-128,128], meaning the a and b values for each pixel could have 256 possible values or 512 between the 2 color spaces. This makes sense because the network is trying to learn the a and b values for each image and then map pixels to those values. After blowing up to 512 channels, the paper suggested

[1] Our CNN performs a softmax activation function with a temperature or T = 0.38 (as recommended by Zhang et al.) since lowering the temperature produces a stronger peaked probability distribution of the a*b values

we scale down the channels to 256 in block 8, up to 313 for our probability distribution for the a*b values, and then down to the 2 a and b channels themselves.

- The objective function recomended by Zhang et al. is essentially a cross entropy loss function multiplied by class weights, where the classes are the potential a*b values of the pixels. It treats the problem of guessing the a*b values of an image as a multinomial classification problem to find the distance between the probability distribution $\widetilde{Z}$ and Z. They found this type of loss function to be more effective than when they compared it to L2[2]:

$$L(\widetilde{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \, log(\widetilde{Z}_{h,w,q})$$

  - Z = $\mathbf{H^{-1}}$(Y), $\widetilde{Z}$ = Probability of a pixel being one of the q possible colors/output of our CNN
  - h = Height of image, w = Width of image, Y = Ground truth image
  - q = Number of colors being considered in colorization, v = Weights to rebalance loss depending on rarity of color
  - T = Temperature of colorization (Zhang et al. found 0.38 to be most effective, so we shall implement this as a constant).

  Colorization of our image will be given by the soft function:

$$\mathbf{H}(Y) = Expected[f_T] = \frac{exp(log(Z)/T)}{\sum_{q} exp(log(Z_q)/T)}]$$

The method, at first, strong enough when first starting out with because Zhang et. al was able to fool a human that the model's result was actually an original, color photo and not some image with guessed colors for about 32% of instances. This is considered pretty high for as far as image colorization papers go. However, I was not able to implement this model exactly as Zhang et al. intended. First off, we were getting errors with running the PyTorch API's implementation of cross entropy loss (nn.CrossEntropyLoss) with our model, so we had to change the loss to mean squared error loss for the sake of comparability to the API. More work will need to be done to mediate this.

The method has a relatively high rate of creating passable, colorized images and is not too complicated so that adjustments can be made and easily implemented into the model. Also, having access to the PyTorch API's neural network containers made the implementation significantly easier as I was able to easily align and experiment with the channel dimensions. One thing that I struggled on, however, was determining an appropriate parameter for our layers (the kernel size, how much padding, how far a stride).

3 Experiments

The dataset I had initially planned on using in the project proposal for training ended up being different from the dataset I used in the final project implementation. The original plan was to use a L*a*b dataset of 25,000 images that I found through the Kaggle database. The layers of the images came in separate files. One file for the L data (grayscale data) and another file for the a *b (the color data). I ran into issues, however, when I was trying to feed this data into our machine learning model. The dataset off of Kaggle was already pre-split into L*a*b channels which made it hard to work with the entire dataset at once since this was too much data for our processors to handle at once. So it was decided to change the dataset to an RGB dataset that we converted to an L*a*b format. It was much simpler using an image converter than to use the original LAB dataset because we did not have to preload every L*a*b image onto our program and could still access it at runtime.

The data came as a set of 41,000 images. I split this into a set of 40,000 for images training and 1,000 images for validating. Since these were RGB images, we had 3 channels of 256x256 pixels that we converted to L*a*b during runtime to save space on the disk and lighten the pre-computational load. I began running our model on the testing data to make sure that the output dimension of the data was

[2]L2 distance was the loss function that I used in my training and in the github respository, however, as I am still experimenting with different loss fucntions

correct and I was able to get some plausible values for the model without expending the time it takes to train over the entirety of the 40,000 images in the training set.

Results:

- I tried to experiment with cross entropy loss, as the paper described and we have listed above in our Model's Structure section, but kept getting errors, so I switched to mean squared error loss and treated the guessing of the a*b color values as regression instead of a multinomial classification problem. This may have been one of the reasons why I was not able to get as good results as the paper, and in the future, I will continue to work on this assignment to resolve the error with cross entropy.

4 Related Work(Optional)

Luke Kyriazi work:

- Luke Kyriazi created an image colorization neural network all from scratch by using a multi-layered, convolutional, neural network interspersed with ResNet layers. The database he used is a 41,000 RGB image file of 256x256 pixels each. I took the database from this paper because I ran into issues with our proposed, Kaggle database, as mentioned
- The images are separated into 2 files, one for training images and one for testing images. Kyriazi also provides multiple utility functions such as a checkpoint saver, a loader for the training data, a visualization function for the images, and a function to convert RGB images to L*a*b, which was very useful to decrease the disk space needed to train our model and as will be discussed later in this paper.
- This resource was very useful for my understanding of how I should use, load, and process the data to run the model. I also utilized his RGB to L*a*b function for the project, which gave the output needed to run and train the model. He makes use of the sk-image library native to python to basically convert from RGB to LAB.

References

Shetty, Shravankumar. *Image Colorization*. (Image Set) Kaggle/Public Domain, 6 Sep 2018. Web. 4 Feb 2021. <https://www.kaggle.com/shravankumar9892/image-colorization>

Zhang R., Isola P., Efros A.A. (2016) Colorful Image Colorization. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9907. Springer, Cham. <https://doi.org/10.1007/978-3-319-46487-9_40>

Wikipedia. *CIELAB color space* <https://en.wikipedia.org/wiki/CIELAB$_c olor_s pace >$

Kyriazi, Luke Melas. Image Colorization with Convolutional Neural Networks, MIT, <lukemelas.github.io/image-colorization.html.>