Black and White Image Recoloration with a Convolutional Neural Network

Ilana Zane

December 18, 2021

1 Introduction

Image recoloration and restoration from black and white photos is an arduous process requiring artistic skill, understanding of color composition, and Photoshop experience. However, there are now several ways to utilize the power of machine learning to recolor images in a fraction of the time. Popular methods of recoloring involve using generative adversarial networks (GAN) or convolutional neural networks (CNN). In this paper, I attempt to create my own version of a CNN to recolor black and white images, and explore how well the network performs when it is provided with images that are missing information. In the following sections I explain my approach and results from the project.

2 Approach

Although a GAN is more powerful, as they are typically the result of two CNNs, single CNNs are still capable of effectively recoloring images based on their ability to extract different features in different layers of the network. In short, a CNN first extracts low level features such as curves and edges, and then it continues to extract patterns and higher level details. Once the CNN has these different layers of varying detail, it is able to combine these layers to create a full image.

2.1 Model

I based my model off of [2] to serve as a starting point, and made several modifications to create a model that better suited the project. Training and data generation parameters were edited so the model could run on my laptop and through Google Colab. Before training, the images are preprocessed by converting them from RGB to the Lab color space, since Lab is a more accurate color space to describe the human perception of color. The L channel of the image, which describes lightness through gray-scale, is isolated and used as input for the model. The model consists of 12 convolutional layers, created through the Tensorflow platform and Keras library, that take images of size 256 x 256. In order to extract features, the model has a kernel size of 3x3 and a stride of 2. This means that images are analyzed by a 3x3 window to see only nine pixels at a time and the stride parameter dictates that the window moves by 2 pixels each time. This essentially leaves us with an image that is half the width and height of the original. Towards the end of the process, the network upsamples the image in order to increase the dimensions of the image to what it originally was.

2.2 Training and Testing Datasets

The training data-set consists of 10 256x256 colored images of faces that were converted from RBG to Lab. Since this is a relatively small number of images for what is usually needed to properly train a model, a data generator was used to make 50 unique copies of each original training image by rotating, flipping, and zooming in on them. A main reason for doing this instead of just using thousands of raw images is to both save time training the model as well as efficiently use computational

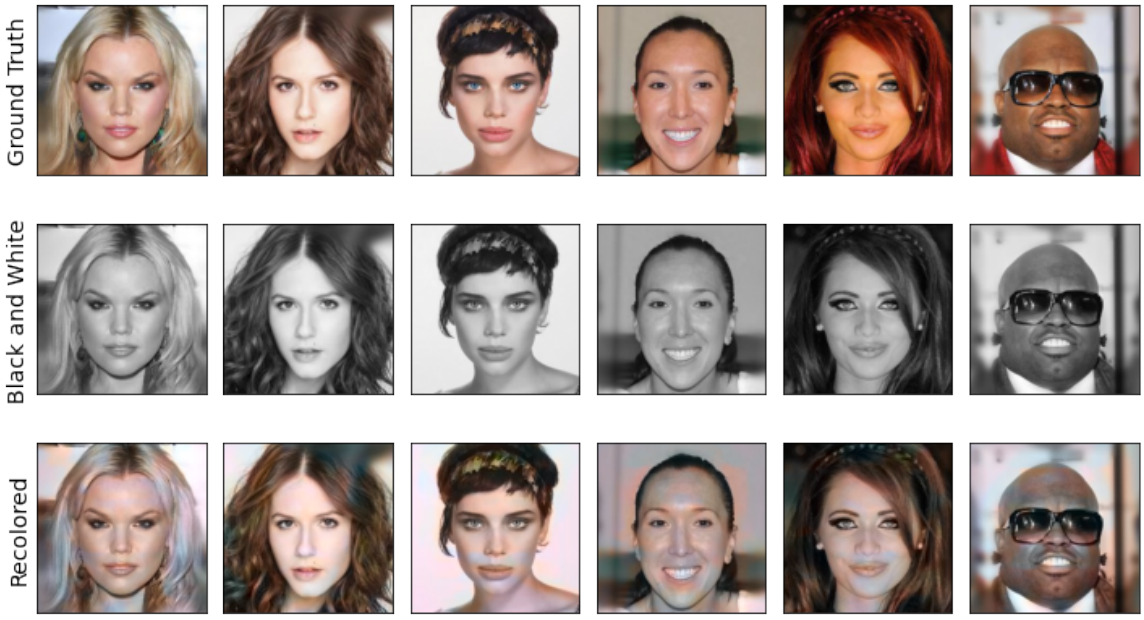

Figure 1: A set of ground truth images (top row) that have been converted to black and white (middle row), and then recolored by the network (bottom row).

resources. Since I am running this network directly on my Mac Book laptop, there is only so much computational power available for me to use. The model never sees the original training data and uses this generated data. The testing data-set consisted of 15 images of faces from the CelebA-HQ data set [1] found on Kaggle.

2.3 Training the Model

In terms of training the model, there were a variety of parameters to consider. Since the number of images being used was relatively small (even after the data generation), it was important to then run a large number of steps per epoch in order for the model to see enough of the training data. I experimented with several different parameters and finally settled on 1,000 steps per epoch with 1 epoch in order to achieve decent enough results, while keeping the run-time at a reasonable length ($\sim$1 hr of training). With only a few steps per epoch, I found that the recolored images were predominantly one color (green, red, blue, purple), so it seemed as though this needed to be kept relatively high in order to have the model recolor the images closely to their original ones.

3 Results

3.1 Recoloration Performance

The results of the model can be seen in Fig. 1, where six black and white faces have been recolored. The first and second rows show the ground truth and black and white image, respectively, while the third row shows the recolored image. Faces were chosen as the testing since the model was trained exclusively on faces, although in principle any 256x256 black and white image could be used.

From these results one of the first things that can be noticed is that the recolored images are much less vibrant than the originals. The model seems to be biased towards the color blue, as there are extra traces of the color in almost all of the images where it doesn't belong. In addition, the model seems to struggle with recoloring people with blonde hair as seen by the first column, and also has trouble picking up bright red colors, which is particularly noticeable in the fifth column. In general, however, the model was able to capture skin tones fairly well and able to match most of the original image.

Part of the reason for this may have to do with the fact that the training set consisted of mostly white women with darker hair. The images in the training set were portraits that included the subjects' bodies in various poses, with a lot of different background colors. The range of colors might contribute to the overall gray/brown tint that most of the test images have.

While it was not explicitly tested, it would be interesting to see how well the model performs on images that contain things other than faces.

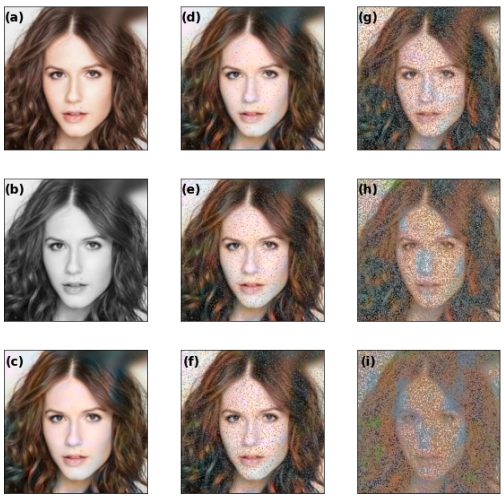

Figure 2: A (a) ground truth, (b) black and white, and (c) recolored image, as well as versions that were recolored by the model after having 1% (d), 5% (e), 10% (f), 25% (g), 50% (h), and 75% (i) of the black and white version removed.

3.2 Loss of Information

Once the model was trained and able to adequately recolor black and white images, I decided to to extend the project to see how well the model can recolor an image with missing information.

Different percentages of random pixels were removed from the black and white images and then recolored. Figure 2 shows the results of recoloring an image that is missing 1%, 5%, 10%, 25%, 50% and 75% of pixels selected at random. The pixels removed from the image were replaced with gray pixels i.e had a pixel value of 0.5 on a scale of 0-1. Replacing missing information with gray pixels replaced lead to results that were more moderate compared to replacing the pixels with white (0.0) or black (1.0) pixels. Using white pixels created images that became increasingly more yellow and black pixels rendered the last image (75%) almost completely black. When the image is recolored with gray pixels, the outline of the original image can be clearly seen, and a majority of the color (brown hair) is recolored correctly.

For two images x, the original image and y, the recolored image with missing information, with N pixels each, the overall performance of the model was based off of the average distances between to two corresponding pixel values of the original image and the recolored one. That is, the score of the model was computed as

$$score = \frac{1}{N}\sum_{j=0}^{N} dist_j \tag{1}$$

where $dist_j$ is the distance between pixel values across the two images. Since rgb pixel values can range anywhere from 0 to 1, recolored images that return average scores closer to 0 when compared to the original image means that the model did a good job recoloring. Eq. 1 can then be used to look at the overall performance of the model as a function of percent lost, as seen in Fig. 3.

It can been see from Fig. 3 that percent loss gradually increases as average pixel difference increases. This makes sense because when the there is less missing information I expect the average pixel difference to be much lower. Otherwise, an image with most of its information

missing will be recolored less accurately with our current model.

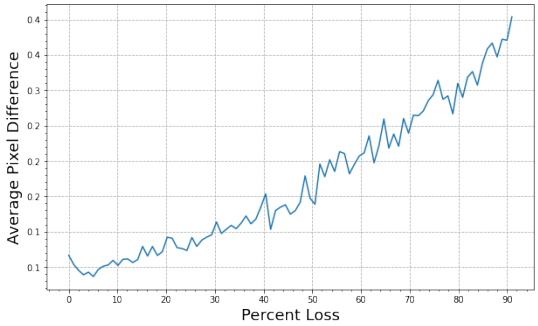

Figure 3: Performance of the model as a function of the percent of pixels removed from the original black and white image.

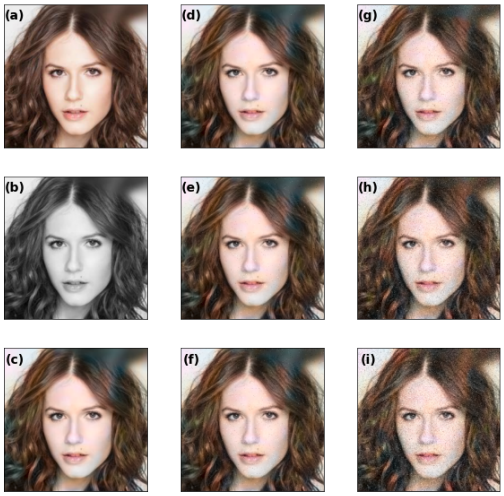

Figure 4: A (a) ground truth, (b) black and white, and (c) recolored image, as well as versions that were recolored by the model after having 1% (d), 5% (e), 10% (f), 25% (g), 50% (h), and 75% (i) of the black and white version tampered with.

3.3 Random Noise

One of the issues with the methodology of the previous section was that entire , and in addition, every pixel was treated the same by being converted to a purely black one. In real life scenarios, images may contain random forms of noise that are non-uniform. One can imagine that during some scanning process or digitization of a physical black and white image, some random noise is introduced. So instead of completely removing information from an image, it may be more beneficial and instructive to introduce a sort-of noise into the images, randomly tampered with the pixel values in a non-uniform way. As we will see, this methodology turns out to improve the performance of the model in certain situations. An example of this technique can be seen in Figure 4, where 1%, 5%, 10%, 25%, 50% and 75% of the pixels were randomly selected and tampered with by adding some noise. Since the pixel values lie within a range of 0-1, random float values within this range were added to the selected pixels in order to simulate noise.

Similar to the previous section, an analysis of the average score of the model as a function of percent loss can be done. As expected in the previous situation, the average score/pixel difference increases as the percent of information tampered with increases, however, this time there were a few images where introducing some noise actually improved the performance of the model. This behavior can be seen in Figure 5, which shows the performance of the image in Figure 4.

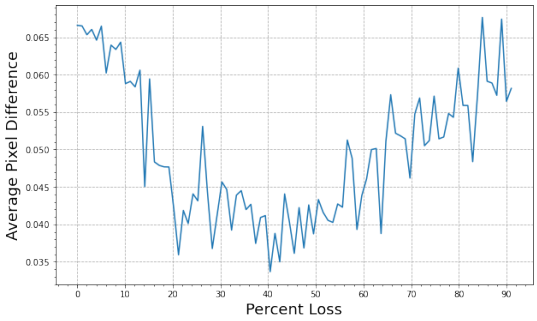

Figure 5: Performance of the model as a function of the percent of pixels tampered with from the original black and white image.

What is interesting to note is that for this image, it seems as though it is beneficial to add some random noise to the black and white image before recoloring to get an overall higher final score. After looking at the recolored images given in Figure 4, it does look like some of the more noisy images retain some of the colors better than the less noisy ones. For example, Figure 4c at 0% noise seems to have some unnecessary blue shades appearing at the edges of the woman's hair, yet the higher percentage ones seen in Figure 4(g-i) seem to lack this.

4 Conclusion

Although the model works, it can be improved in several ways. In hindsight, I would have made the training set have images from the CelebA-HQ data set. The training data that I used is different enough from the testing data that it might have affected the performance of the model. I would also make sure that the training set consists of faces that have various skin tones because currently, the training set is bias by containing only images of white women. We can see from the results that although the model is able to gauge the lightness/darkness of skin tone, the recoloring of the last image is not as clear as the first few recolorations. Although the model may perform better with images that are of uniform color, it is important to make sure that the model is improved to work well with a diverse data set to prevent any bias.

In terms of calculating the score for the models performance, there might be a better way to do it. Currently, I compare total average pixel value between the original image and recolored image. A low average value is indicative of the fact that the model did recolor an image well, however this may not be true. If, for example, an original image has very vibrant colors and the recolored image is potentially recolored very well, but with more muted colors, then the score might be low. This would not be an accurate representation of how the model is performing. To remedy this, I could instead calculate the average pixel difference when the final image is in the Lab color space, instead of RGB as it is now. By doing this, images that are of similar color but at a different level of brightness will not be rewarded with a low score.

In addition, some shortcuts were taken during the training process due to a lack of computing power and proper resources. I used Google Colab to train the model, but it was for 2 hours at most before my sessions were terminated. I saved the weights for the first model that produced a recoloration that was not skewed to one of the a/b channels. If I could I would properly train the model with more data and for a longer period of time in order to produce the best possible results.

5 How to Run Model

My project consists of a Jupyter Notebook with the code and a file called $model_main.h5 that contains saved weights for the best performing mo$

References

[1] Moses Odhiambo. Celeba-hq resized (256x256), 2021.

[2] Emil Wallner. How to colorize black white photos with just 100 lines of neural network code, 2017.