Project 4 - Colorization

Ilana Zane,Beatrice Liang-Gilman, Will Bidle, Abinaya Sivakumar

May 4, 2020

1 Basic Coloring Agent

Figure 1: Original image

1.1 Methodology

For our k-means model we first chose five random coordinates to act as our centroids. We then grouped each pixel with a centroid by using the nearest euclidean distance:

$$Distance = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1}$$

Here, x_i and y_i represent the i^{th} components for the vectors x and y, where x is a pixel, and y is a centroid. In this case we take n = 3 since we have 3 dimensional vectors (r,g,b). Whenever a pixel found its nearest centroid, it was classified as belonging to that centroid. Then, for each cluster we calculated the average RGB values, which became our new centroids. We then continuously reclassified all of our pixels to the new centroids and recalculated our centroids until each was within a 5x5x5 pixel volume of its respective

average (centroid's blue value is within five pixels of cluster's blue average, and same for green and red). At this point the program would stop and take these values as the final centroids. To recolor the right side, each 3x3 set of pixels was compared with the six most similar 3x3 sets of pixels on the left side. We defined similar to mean 'most similar in color,' and calculated this by using a nine dimensional distance formula to find the distance between the nine grayscale values from the test and training set. To do this we used Equation (1), where n = 9. Once the six most similar patches were found, we looked at the representative color of these patches and recolored the right middle pixel based off of the majority color. The representative color was the color of the centroid that the middle pixel was mapped to. In the event of a tie, we recolored the right based off of which patch was the most similar.

1.2 Analysis

Figure 2: Right side recolored with k-means

1.3 Result

The result is actually really good considering the limited data the model was given to train with. As is seen in the picture, the colors in the right half correspond well to the ones in the left half. The major difference comes in the finer details of the image, as can be seen in some of the sky and trees. Some of these errors are understandable, as it picks colors that are close to what is expected (i.e. picking purple red instead of purple).

1.4 Improvements

There are a few major things that could be improved, the first being the variety of colors. This could easily be done by using more centroids. Limiting the model to use only 5 centroids results in a patchy final image, and a lot of the finer details are left out. Even

though several main parts of the picture were recovered (i.e. the dark land/trees and the different shades in the sky), using more centroids would allow us to recover some of the original complexity of the image. The basic agent's runtime was another area that could have been improved. The computation of centroids itself was fairly quick, however the runtime steadily increased during the recoloration process. In general, comparing each 3x3 test patch to every single 3x3 training patch is computationally expensive, especially for reasonably sized images (i.e. 500 x 400). We decided to reduce the time it took by using a percentage of randomly chosen batches from the training data to compare to each test patch, instead of using the entire data set. This definitely helped to make the algorithm run faster, although it came at the expense of the quality of the final result. The smaller the training used for comparison, the worse the resulting image quality. When moving to the advanced agent, we realized we would need to use a similar strategy, in order to appropriately choose how much data we were feeding in and how many iterations we were running on our data. In implementing our advanced agent, we would like to see smoother edges, as well as a larger variety of colors.

1.5 Numerical and Visual Satisfaction

The image is visually satisfying. The colors look appropriate, and the trees in the image stand out accordingly. In addition, the colors on the right side mostly line up with those on the left side. For instance, the orange and tan skylines, as well as the color of the water are consistent on both sides of the picture. The picture is also numerically satisfying. To compare the pictures numerically, we found the sum of the squared differences to find the mean squared difference, as shown in Equation 1, above, with n = 3. As can be seen in Figure 3 below, which plots about 10% of the data (10,000 randomly chosen pixels), the average distance between corresponding pixels in the basic agent and original image was about 44.31. This leads to an average deviation from each R, G, and B value to be about 4, which is a relatively small difference.

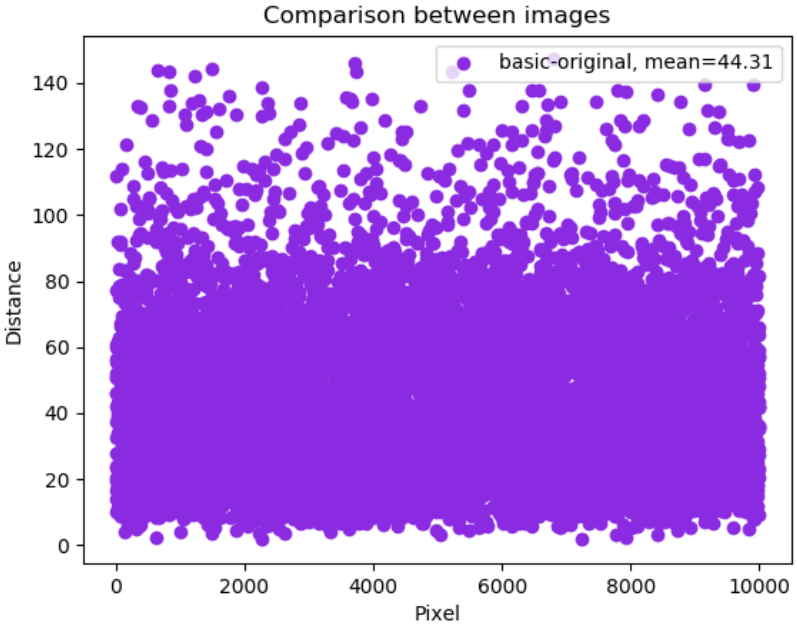

Figure 3

2 Improved Coloring Agent

Figure 4: Right side recolored with neural network

2.1 Methodology

In order to create an improved agent that performs better than our basic agent, we decided to use a neural network. Just as before, we used the left side of the image as training data and the right side of the image as testing data, so that the left side of our final result was the same as the original picture, while the right side was the picture generated by the neural network. Our neural network consisted of three layers: the input layer, one hidden layer, and an output layer. Our input space consisted of all the patches of 3x3 pixels from the grayscale image. Thus, our neural network's input layer consisted of nine nodes, where each node was one of the grayscale values from a 3x3 patch. These nodes are normalized between 0-1 by dividing each value by 255. Our output layer was then 3 nodes, where each node was a red, green, or blue value corresponding to the middle pixel in the 3x3 patch. The output space was made up of all possible combinations of red, green, and blue values. Each of these values was originally normalized to be between 0 and 1, however we then multiplied all the outputs by 255 to represent colors within the range of 0 to 255. The model space was composed of a hidden layer with 5 nodes. We chose to have 5 nodes in our hidden layer, as this was the suggested amount of nodes. This ended up being an appropriate amount, as we didn't have too many inputs, but we also didn't want to miss important aspects of the data. We also only used 1 hidden layer (again recommended) to keep things simple. The cost function was found by summing up the squares of the differences between the neural network's output nodes and the actual values from the original image. Thus, if each output node is indexed by i, and given an output layer and an actual value layer y, the cost function is:

$$Cost = L = \sum (output_i - y_i)^2 \tag{2}$$

Given 9 input nodes, we needed 45 different weights in order to calculate the 5 nodes in the hidden layer. We then needed 15 more weights in order to calculate the 3 nodes in the output layer. Each layer can be found given the nodes in the previous layer as well as the weights that connect them together, as seen by the equations below:

$$\underline{Out}^0 = (1, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) \tag{3}$$

$$\underline{Out}^1 = (1, \sigma(\underline{w}^1 \cdot \underline{Out}0), \sigma(\underline{w}^w \cdot \underline{Out}^0), \sigma(\underline{w}^3 \cdot \underline{Out}^0), \sigma(\underline{w}^4 \cdot \underline{Out}^0), \sigma(\underline{w}^5 \cdot \underline{Out}^0)) \tag{4}$$

$$\underline{Out}^2 = (\sigma(\underline{w}^R \cdot \underline{Out}^1), \sigma(\underline{w}^G \cdot \underline{Out}^1), \sigma(\underline{w}^B \cdot \underline{Out}^1)) \tag{5}$$

Where Equations (3), (4), and (5) represent the input, hidden, and output layers respectively. The sigmoid function was used for each activation. To use gradient descent, we first needed to know how to calculate each partial derivative of the loss function with respect to the individual weights. The notation used was inspired from Professor Cowan's *ExampleNetwork* notes:

$$\frac{\partial L}{\partial w_k^C} = \frac{\partial L}{\partial Out_C^2} \sigma'(\underline{w}^C \cdot \underline{Out}^1) Out_k^1 \tag{6}$$

$$\frac{\partial L}{\partial w_i^k} = \frac{\partial L}{\partial Out_k^1} \sigma'(\underline{w}^k \cdot \underline{Out}^0) Out_i^0 \tag{7}$$

Where the Equation 6 describes the weight derivatives between the hidden layer and output layer, while Equation 7 describes the weight derivatives between the input layer and hidden layer. The middle and last terms of each equation correspond to the derivative of the activation function, while the first terms in each equation are found by the following equations:

$$\frac{\partial L}{\partial Out_C^2} = 2(Out_C^2 - y_C) \tag{8}$$

$$\frac{\partial L}{\partial Out_k^1} = \sum_{C=0}^{2} \frac{\partial L}{\partial Out_C^2} \sigma'(\underline{w}^C \cdot \underline{Out}^1) w_k^C \tag{9}$$

Where Equation 8 describes how the loss function is affected by the output nodes, and Equation 9 describes how the loss function is affected by the output and hidden layer nodes. We utilized Equations (6) - (9) to perform gradient descent. To start off, we randomly generated all 60 weights as values between 0 and 1. To keep things simple, we also decided to set the bias to zero. A non-zero bias would require us to calculate a few more derivatives, and generally would make the result more accurate. However, in this case, we are dealing with a simple situation, and can solve the problem without considering it. To preprocess our data, we added all of the patches from the left side of the image into a list. The information in each patch included the 9 grayscale values, along with the coordinates of the middle pixel of the patch. Then, in order to train the data, we shuffled the order

of all of these patches. We also split all of the patches into a predetermined number of groups (we used 200 groups for our image), so that we could train the neural network on a group of patches a few times, before moving onto the next group. We found that this process of stochastic gradient descent made our final result more accurate, as the neural network was able to learn and adjust accordingly to different groups of data. To train our neural network, we calculated the activation nodes based on our starting weights for one patch of data. We then used gradient descent to find how much these weights should be changed according to that patch. Keeping note of these derivatives, we then repeated this process for all the remaining patches in the group. We then adjusted our weight matrices based on the average of all of our derivatives, and repeated this whole process using the same group of data, but the new weights, for 100 iterations. We then repeated this entire process for the remainder of the groups of data. Once the network was trained, we were able to feed our right side patches in, and get relatively accurate (r,g,b) values in return to recolor our right-side image. In order to avoid overfitting, we tested our neural network using different numbers of groups and iterations of times we adjusted the weight matrices for a given batch of data. We found that if either of these values were too high, our image would either stay the same or get worse, so for our image size, we found the number of groups and iterations that produced the best result.

2.2 Analysis

We can see from Figure 4 that our neural network is a close representation to the original image, and is much clearer than the basic agent. All the proper shapes, such as the trees and land, can be clearly seen, and the ratio of shades of color to each other is also clear. However, there are a few colors missing in the right half of the image that are present in the left side. This could come from lack of data, having only trained on the left half of the picture. The network may also have clumped the dark reds and purples together, for example, and lost some valuable information about the original picture. What the network was able to pick up on very well was the color ratios between the darker trees and land to the brighter colors of the setting sun.

2.3 Comparison

We can qualify comparing the performances of the basic and improved agent just by looking at our final results. We can see that the image produced from our neural network looks much smoother and realistic. Where the basic agent produces grainy shapes and colors, the neural network creates a much more gradual transition between colors. While the neural network does not match the exact colors of the left side, it does a much better job at creating a realistic looking image. The reconstructed right side matches the smooth transitions of colors, and captures the finer details missing in the basic agent. Our basic agent and advanced agent provide different results because each agent is attempting to optimize their results based on different parameters. Our basic agent is able to maintain a continuation of color because the whole image is limited to five representative colors. Meanwhile, our advanced agent does not have this limitation, and is open to every red, green, and blue value between 0 and 255. To quantify our results, we computed a difference between the neural network's result and the original image, and compared it to Figure 3. The results can be seen in Figure 5 below.

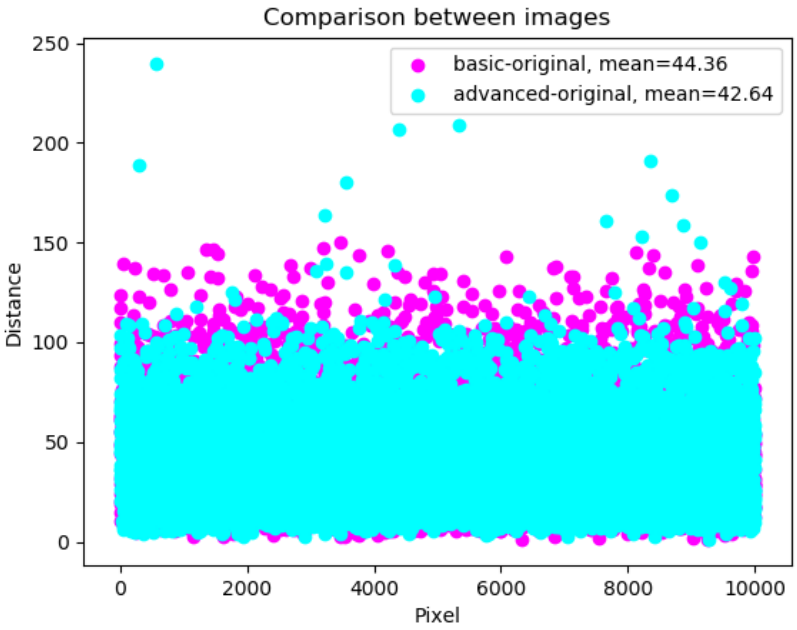

Figure 5

When comparing the performances of both agents, it was important to consider how we could compare them equally. Since the neural network is trying to recreate the original image, it makes sense to compare it directly with the original image. We also compared the basic agent to the original image, to see how our advanced agent performed compared to the basic agent. The average distance between pixels in the advanced agent was 42.64, which is slightly less than the average for the basic agent. While this does show improvement, it appears that there is not as much improvement as we would have hoped for. However, we noticed that there are many more outliers reaching larger distances in the advanced agent, skewing the average to be higher. This makes sense as the neural network was not trying to classify each grayscale pixel to a corresponding color like the basic agent, but rather trying to find the pattern from grayscale value to (r,g,b) values. Tying into this, the basic agent is not trying to recreate the original image, since it is only using representative colors. Therefore, while our analysis made a fair comparison in order to determine whether the advanced agent did in fact do a better job than the basic agent, it would be unfair to analyze the basic agent by itself by comparing it to the original image. In that case, it would then make sense to look at the difference between the basic agent's results and the right half of the image recolored in terms of representative colors. By seeing which agent came closer to recreating its expected right half, one would better be able to analyze each agent on its own. The reason our advanced agent performs better than our basic agent is because it is actually trying to learn from its data instead of finding similarity in the data. The basic agent will find which pixels in training are most similar to the test, and recolor based on this information. This leads to the basic agent trying to recolor the right side to match the left side as closely as possible. Even though the final product is not a perfect recoloration of the original image, the right side is a nice continuation of the information provided from the left. While the advanced agent is also using information from the left half of the image, it won't be comparing the right and left sides at all. The neural network attempts to learn what it can about which colors make sense given a set of grayscale values. This leads to a picture that actually learned from its data, with smoother color transitions

and finer detail recognition.

2.4 Improvements

The final results of both agents could certainly be improved with more time and resources. In general, instead of training on only one half of a picture, it would be nice to expose each strategy to several different images, allowing a broad range of data. The k-means algorithm would benefit from using more of the representative colors (as explained above), as well as looking at more similar patches when recoloring. The more similar training patches we compare a given test patch to, the more likely we are to find the true color of that central pixel. As for the advanced agent, there is a lot of wiggle room with experimentation. It might be useful to use another hidden layer, allowing the network to pick out some of the finer details missing from our result. It would also be interesting to see how different activation functions and including a bias would impact the image quality, as they might allow the network to better fit the data it was given. Most of what we had utilized was very basic (partly due to our limited computing power), but powerful enough to yield better results than our basic k-means algorithm. Another possibility would be to change the input space of our model from looking at only nine pixels, to looking at an entire image. A majority of the results coming from a neural network come from the data it was trained on, so providing our network with more data would certainly improve its performance.