AI Project 4: Colorization

Daniel Ying dty16 Sec: 440:06, Kyle VanWageninge kjv48 Sec: 440:05

May 24, 2021

Submitter:

Honor Code:

I abide to the rules laid in the Project 4: Colorization description and I have not used anyone else's work for the project, and my work is only my own and my group's.

I acknowledge and accept the Honor Code and rules of Project 4.

Signed: Daniel Ying dty16, Kyle VanWageninge kjv48

Workload:

Daniel Ying: Formatted the report, wrote the report in latex code. Compiled the results and answers onto the report.

Kyle VanWageninge: Implemented the basic agent using K-means clustering. Gathered the results for the basic agent.

Together: Researched using lectures on how to implement the improved agent. Answered the questions in the write up.

Problem 1

Basic Agent using K-means clustering:

To start with the basic agent we converted our image into a black and white image, by adding each pixel's RGB value and dividing that by 3. Then we started with the k-means clustering on the left half of the original image. To do this, we have set 5 base centers (by using the 5 most common colors in images), this would give us a good starting center to calculate distances for the clusters. After we trained the data 29 times, we were able to get an accurate cluster center for the 5 clusters. Next, we recolored the left half of the image into those 5 different colors and the output would be seen in the figures below.

Once the training set was complete, we moved onto the testing set by using the black and white images. This was done by taking the right half of the black and white image, flattening the 3x3 pixel patch and identifying the 6 most similar patches in the training data. Once these patches were found, the majority center pixel in a single cluster would be mapped to the location of the middle black and white patch pixel in the original image. The result is shown in the figure as the right half of the re-colored image.

The new re-colored picture overall looked very different from the original image. This is because the re-colored picture only used 5 colors. These 5 colors are also averages of the clusters which means that the degree of brightness of the color were also changed to just one brightness. Thus making the image to look somewhat off. The re-colored test's side also is a bit more off than its left side counterpart. This is caused by the usage of finding similar distances in patches. When comparing patches, there is a possibility that there is no majority cluster, even if we were using the most similar distance. Thus resulting in the pixel being colored incorrectly. As one can see, in the testing data for both pictures, there are some pixels that just don't quite fit.

The final test produced a pretty good result, in fact, we can still see what the original image was. For image 1, an orange Lamborghini in a garage and for image 2 a formula 1 car. Visually they are not as appealing to the eyes as the original, but they somewhat added a rather beautiful element to the image, one that is similar to those observed in paintings.

Figure 1: 150x150 image.

Figure 2: Car in bw image.

Figure 3: Left (Train) vs. Right(Test).

Figure 4: 250x250 F1 car.

Figure 5: black and white F1 car.

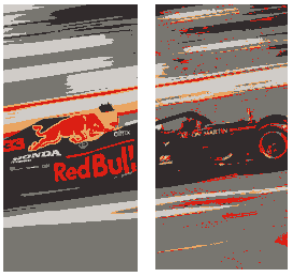

Figure 6: Left(Train) and Right(Test).

*

Problem 2

Bonus:

1) Instead of doing a 5-nearest neighbor based color selection, what is the best number of representative colors to pick for your data? How did you arrive at that number? Justify yourself and be thorough.

2) Why is a linear model a bad idea?

Answer:

1) The number of representative colors that should be picked for our data should be 20. As shown in the figure below, the color wheel image shows that there are 20 colors on the list consisting of 5 main colors with varying hues. One of the main issues for the basic agent was that all the hues got averaged into one color instead of having 3 different averages (bright color, darker color, and mid color average of all the colors seen). With these 20 cluster center points, this would give the image all the base colors, it would need to recreate the original image with extra hue varieties, thus producing better images than the basic agent.

2) Linear models would not work for the improved agent because the cluster of colors would not be in a linear fashion. They would vary in location using the different RGB values, thus a linear line would not be able to be taken to the average of any values. So using a parametric model would more accurately represent the dips and pulls in the values.

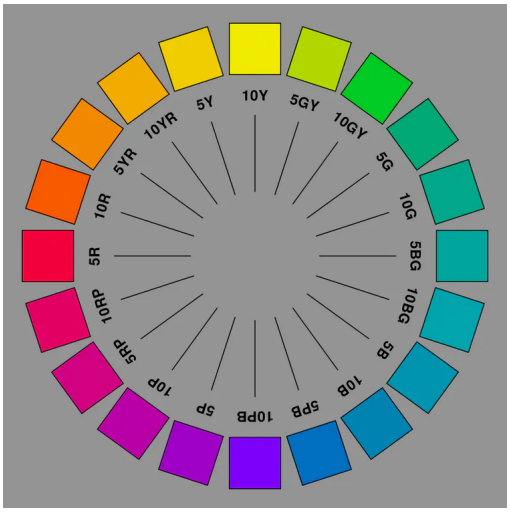

Figure 7: Color Set Wheel with 20 color varieties.

Problem 3

Improved Agent:

ACRONYM: C.A.T (Color Alternating Tester)

Though we did not get to implement our improved agent into a program due to the difficulty the agent, we did have implementations we were trying:

A logistic regression solution uses the sigmoid function which would take in a black and white patch of test data. By using the equation:

$$\nabla wLoss(fw(x), y) = [fw(x)y]x$$

We would be able to run the program until the loss was to minimal. The training used for this solution would be the stochastic gradient descent which would include a learning rate depending on if we are trying to find the red, green, or blue of a value. With 2 for red, 4 for green, and 3 for blue, we would choose the data point that we want to find the value and assign to it a test output that would compare what patches are similar. This would then be run with the training equation:

$$w(k+1) = w(k) - \alpha[f_{wk}(x) - y]x$$

.

We would repeat this for 10000 times or more to get a solid color output for each color. To avoid over-fitting, we would include cross-validation on the training data.

To check if the improved agent worked better than the basic agent, we would compare how close each agent got to the original photo. To make this more fair, we can average each red, green, blue value of the pixel and see which photo came the closest. Then give a final percentage at the end of the photo that came closer to the original photo. This would be fair because there exists a possibility which the photo had a few pixels that were completely off while the basic agent got a consistent accuracy with each pixel, giving the best comparisons.

If given more time and resources, we would like to make our agent use a neural network. To implement this, we would set each layer to recognize a certain type of color depending on the most similar patches. Then for each hidden layer piece, set them together based on the varying brightness and hues of their color.