Computer Vision Project 2

Joel Yuhas

March 2019

1 Introduction

Computer vision relies on the ability to transform and analyze images in order to better understand them. This project involved utilizing several techniques that involved k-means, histogram analysis, image matching, texture matching, Garbo filters and more. These features were used in order to generate and compare image histograms to one another, generate and compare image texture features to one another, and also generate and compare images that had been analyzed with a gabor filter bank. Python scripts were created that utilized these methodologies and produced the results that were analyzed. Overall the project ended up being a success and all of the results were in the expected ranges.

2 Color Image Segmentation Using K-Means

Methods/Results Histograms are a very useful tool that can be used to help analyze images. The first section of this project read a color image and created a histogram for the red, blue and green channel, as well as a final concatenated histogram. The histogram was set with a bit size of 384. The following figures shows an image, the R,G,B histograms, as well as the concatenated histogram.

Figure 1: Original Image

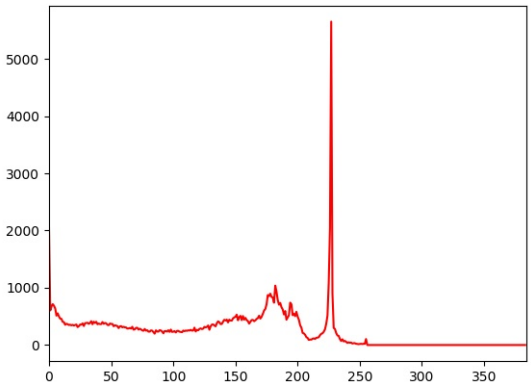

Figure 2: Red Histogram

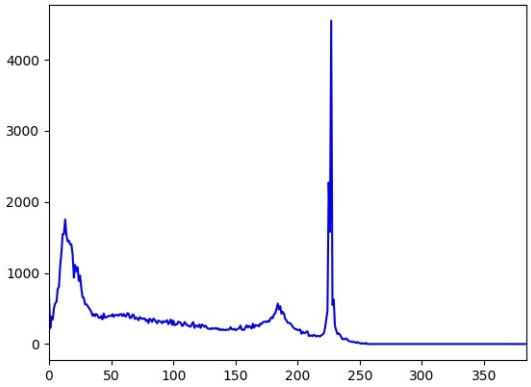

Figure 3: Blue Histogram

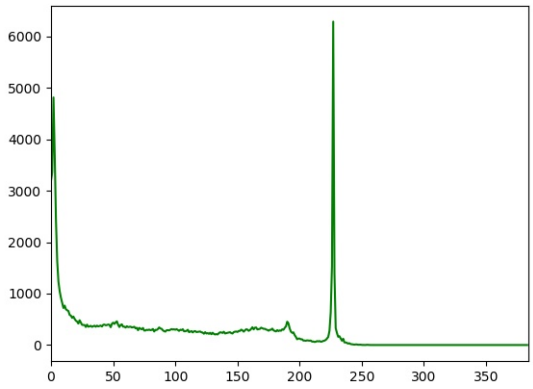

Figure 4: Green Histogram

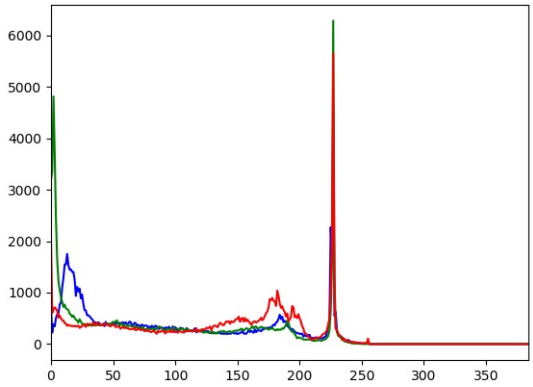

Figure 5: Concatenated Histogram

Each histogram represents the intensity value of the pixel on the x-axis, and the frequency of that pixel on the y axis. Each histogram, R, G, and B shows the intensity and frequency for the red, blue, and green values respectively

Next, K-Means were implemented. K-means is a process where a number of centroids are selected. These centroids are random points. All of the points in the images histograms that are closer to one centroid are a part of that centroids cluster. Once all of the points are assigned a cluster, the center of those points is calculated by taking the average of all of their euclidean distances. The centroid for that cluster is then placed on that cernter point, and the process is ran again. This technique can be used to find grouped together sets of data. Specifically

for this project, it was used to group together similar colors. Once the centroids converged or the max iterations were ran, the centroid value was set to be the color value for the specified clusters.

For this project, a k-means algorithm was written. An approach was taken where two methods were written, one that handled a grey scale, while another handled a color image in 3D color space. First the grey-scale image was done. The figures below shows the original color image, grey scale image, and then segmented image. This image was segmented at k=3. Notice how there are only 3 shades of grey in the image as a result from the k-mean process.

Figure 6: Original Image

Figure 7: Grey Scale Image

Figure 8: K-Means

This process was repeated, this time with a small k value and a large k value. The figures below show the results. Notice how the larger the k value, the more colors and segmentation's there are. This is exactly what adding more k values is suppose to do

Figure 9: K-Means Small Value of 2 Black and White

Figure 10: K-means Large Value of 10 Black and White

After the grey-scale image was segmented, the k-means process was applied to a color image. This was a little more tricky, since instead of dealing with only one set of values, now there were 3 sets of values (R,G,B). This meant that the centroids were now located in 3D space, which makes the process more computationally heavy. The figures below show the original color image, as well as the results from the color k-means algorithm using the same k values.

Figure 11: K-means Small Value of 2 Color

Figure 12: K-means Large Value of 10 Color

As mentioned earlier, the larger the k value, the more clusters there are. In this context, that means there will be more colors in the final image. On a side note, it should be noted that the code that was created was set to run 50 iterates every time. It was found thorough testing that this gave the most consistent and overall best results.

3 Image Retrieval Using Color Histogram Features

Methods/Results

The next section used the concept of histogram intersection to match similar images to one another. The concept works as follows. A data base was provided with dozens of images. Some of these images belonged to a certain class, which may have been a bus, landscape, animal, etc. A python program was written that took the histogram of every single one of the images and saved in memory. Then the user selected a target image and the program would calculate the histogram intersections between the target image and all the images in the database.

The idea of histogram intersection is to go through all the values in both histograms, and take the minimum value in each bin. Theoretically, the largest value at the end of the process meant that the two histograms were incredibly similar, and hence forth, most likely a similar image. However the issue is that this only looks at histograms, which means two very different photos could just so happen to have the same color distribution. The equation below shows the equation that was used for the histogram intersection

$$\text{Hmatch}(h_I, h_M) = \frac{\text{intersection}(h_I, h_M)}{\sum_{j=1}^{K} h_M[j]}$$

Figure 13: Histogram Intersection Equation

The code was ran using a 96 bin quantized color histogram for each image. An image from each class of the database was chosen, and the 4 best matches were recorded. It should be noted that the code prints out names of the 5 best matches since the best match should theoretically be the image itself every time. Only the 4, not including the original image, were recorded. The class of each matching image was recorded in a table. This was repeated again with a different image from each class. The results from both of these runs can be seen in the two tables below.

Query	Match1	Match2	Match3	Match4
Greek	Horse	Greek	Greek	Greek
Bus	Food	Bus	Bus	Flower
Elephant	Elephant	Mountain	Horse	Mountain
Flower	Flower	Flower	Flower	Flower
Horse	Horse	Horse	Horse	Horse
Mountain	Flower	Food	Food	Greek
Food	Food	Flower	Flower	Greek

Table 1: Histogram Results 1

Query	Match1	Match2	Match3	Match4
Greek	Greek	Horse	Flower	Horse
Bus	Bus	Flower	Bus	Bus
Elephant	Horse	Horse	Horse	Elephant
Flower	Flower	Flower	Flower	Flower
Horse	Horse	Horse	Horse	Horse
Mountain	Food	Flower	Food	Horse
Food	Food	Food	Food	Flower

Table 2: Histogram Results 2

After evaluating the data above, the 2 best matches and the worst match was selected and placed in the table below

Figure 14: Best and Worst Histogram Results

As can be seen by the table, the top two retrieval results worked very well while the bottom one did not. These demonstrate how color based image re-

trieval can have very promising results. The two best results show very similar matches, with the first 2 flowers looking almost identical to the first. Since histogram intersection can be a relatively quick process, both to create the histogram and compare it, it can be a powerful tool. How ever, as mentioned previously, since the histogram process only looks at the colors themselves, its very possible that two images could have a similar color distribution but end up looking nothing alike. This is demonstrated in the worst case from the table above where none of the classes matched the target class.

4 Image Retrieval using Texture Features

Methods/Results After histograms were used to compare images, texture features were next on the agenda. This section took on evaluating texture features by using a 9-dimensional texture feature based on Laws filter mask. There were 4 1D Masks that were combined together in 9 different ways to create the 9 2D-mask that were used to create the 9-dimensional texture features. Each filter was ran through a specific image, and then all of the results from that filter were averaged together to obtain a single result. After all the filters were utilized, this created 9 separate, single result values that was considered to be the 9-dimensional texture feature. Since each filter was different in its own way, the idea was that each filter would react that similarly to each pattern that it went through, and since there were 9 of them, these values could be used to identify which other textures were similar to them. The tables below show the 1D mask that were combined

Name	Mask
Level	L5 = [1 4 6 4 1]
Edge	E5 = [-1 -2 0 2 1]
Spot	S5 = [-1 -2 0 2 1]
Ripple	R5 = [1 -4 6 -4 1]

Table 3: 1D Mask

These filters were used against a texture database that was provided. The python code that was written went through and created the 9-dimensional texture feature for ever single image in the data base. Since this was a relatively computationally heavy process, the code would generate the values for each image and save them to a file so they could be read off later. Next, an image from each texture class was selected and compared against the database. The images were compared by taking the Chi-square distance between each feature vector. A small distance meant a closer match. The chi-square distance equation can be seen below.

$$\chi^2(h_i, h_j) = \frac{1}{2}\sum_{m=1}^{K}\frac{[h_i(m) - h_j(m)]^2}{h_i(m) + h_j(m)}$$

Figure 15: Chi Square Equation

The top 4 results were recorded. Similar to the histogram section, the code printed out the top 5 results since the highest was usually the image itself. This was repeated again but with a different image from each class. The class of the image and the class of the match's were recorded in the tables below.

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Plaid
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Bark	Carpet
Plaid	Plaid	Plaid	Plaid	Plaid

Table 4: Texture Results 1

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Carpet	Bark
Wood	Wall	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Bark	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Bark	Carpet
Plaid	Plaid	Pebbles	Plaid	Plaid

Table 5: Texture Results 2

After reviewing the results, similar again to the histogram section, the best 2 cases and the worst case were recorded. This time the images were displayed. The results of the best and worst can be seen below.

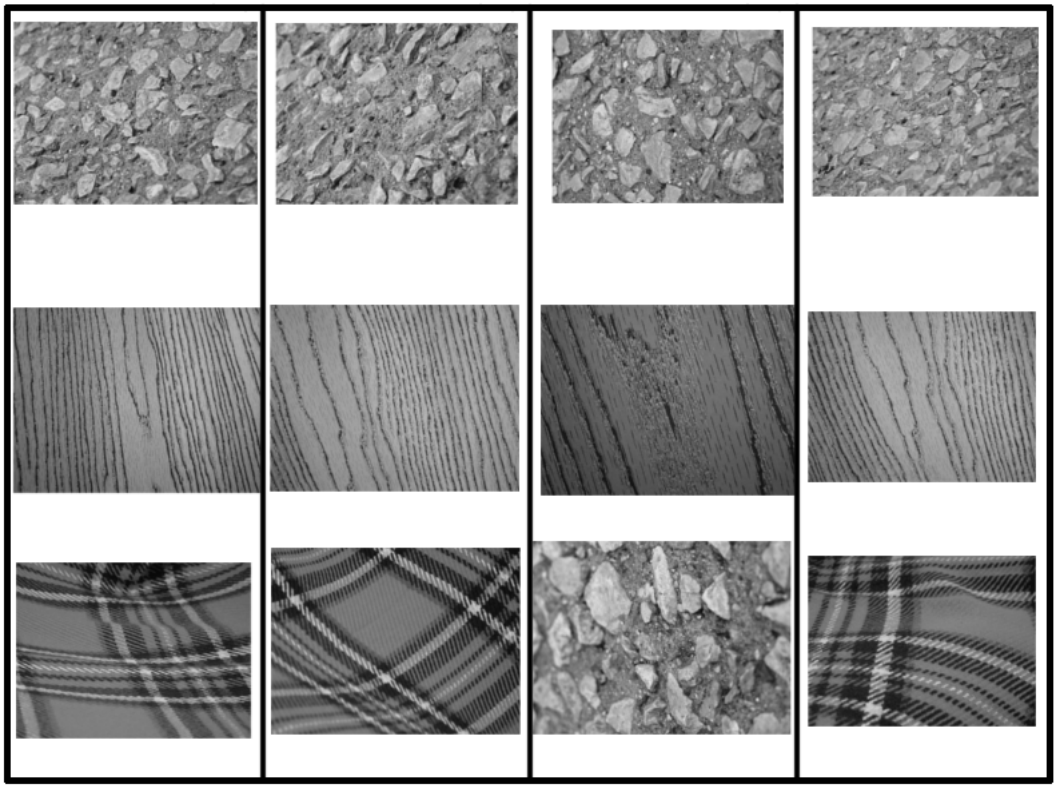

Figure 16: Textures Best and Worst

Sometimes when comparing images, resizing them can improve the retrieval results. To test this theory, three resizing test were ran. One that resized the images to a smaller square at 250 x 250, one to a larger, more vertical rectangle at 800 x 400, and one to a narrow more horizontal rectangle at 200 x 800. The results of the matching classes for each case can be seen below.

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Plaid
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Bark	Carpet
Plaid	Plaid	Plaid	Plaid	Plaid

Table 6: Texture Results Small Square Size

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Plaid
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Bark	Carpet
Plaid	Plaid	Plaid	Plaid	Plaid

Table 7: Texture Results Vertical Stretch Size

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Plaid
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Bark	Carpet
Plaid	Plaid	Plaid	Plaid	Plaid

Table 8: Texture Results Horizontal Stretch Size

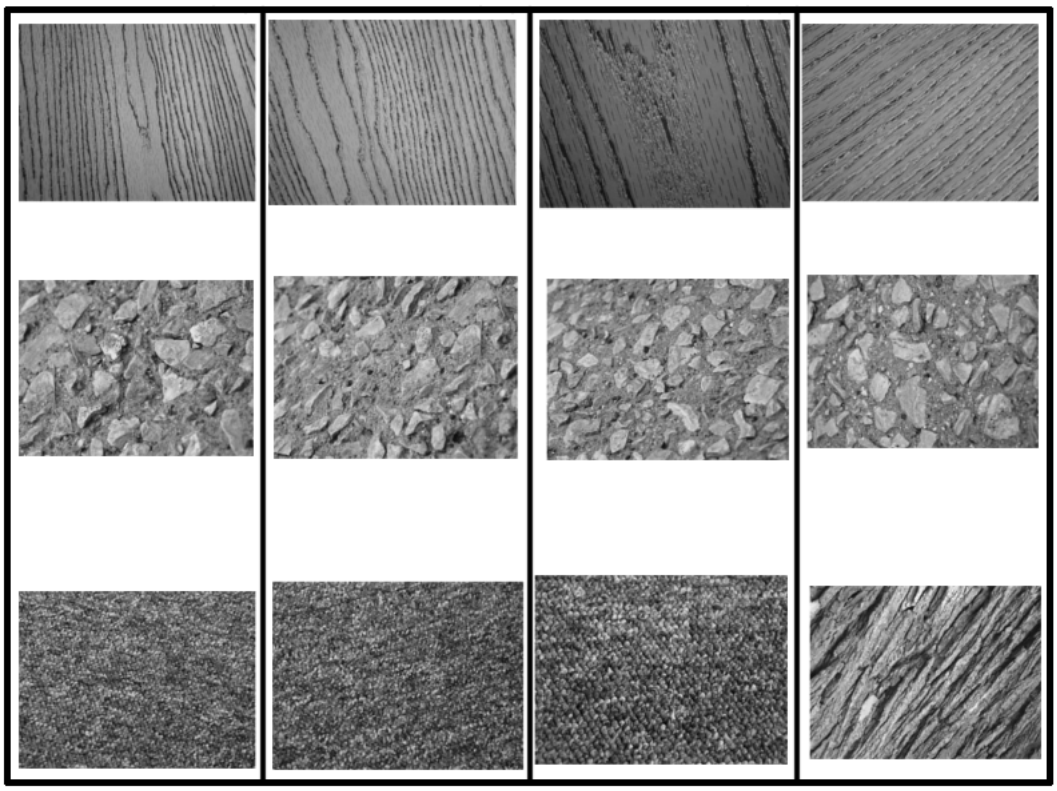

Figure 17: Textures Square Size Best and Worst

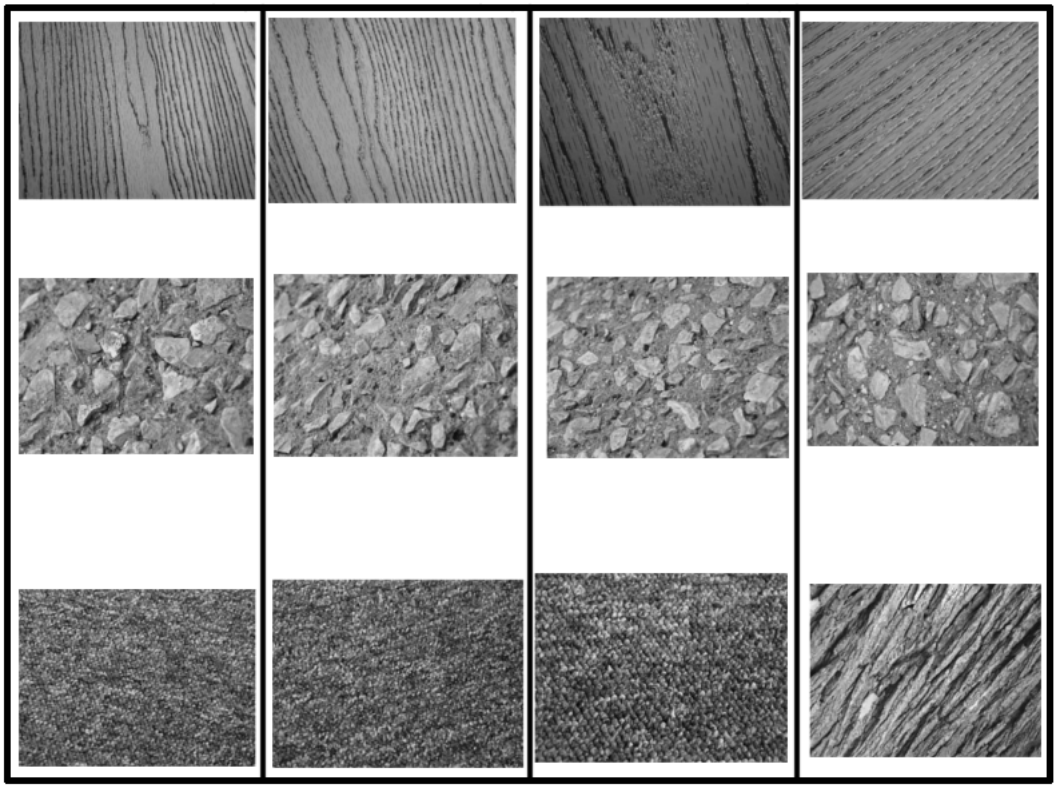

Figure 18: Textures Horizontal Stretch Best and Worst

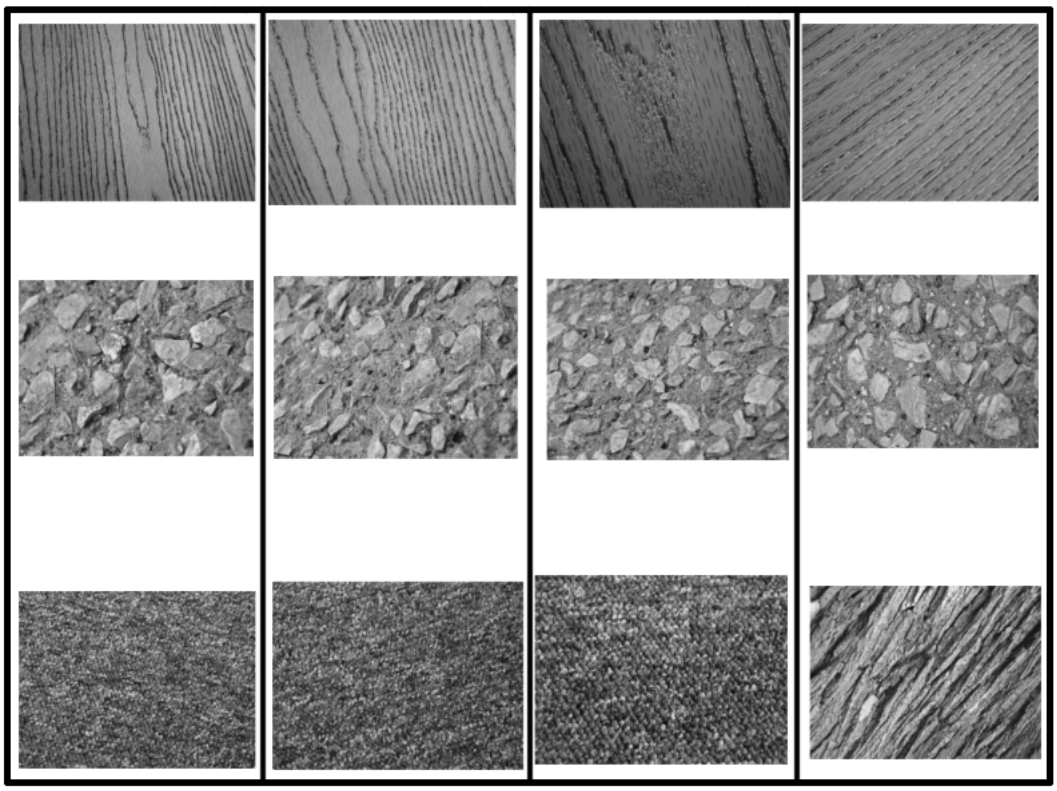

Figure 19: Textures Vertical Stretch Best and Worst

The size difference shows identical results across all of the different sizes. This could be based on two different reasoning's. The first reasoning being that if all of the images are effected the same way, then it could have the same effect across all filters. This would mean that none of the results would change since everything was effected equally. The second could be that the code was not resizing the images properly. However after going into the code it seemed that it was indeed manipulating different values each time.

Texture matching can be a very useful. As seen by the results, the texture matching is often very close at guessing the accurate texture. It also has the added flexibility of being able to add more filters to the texture features, thus increasing the accuracy. However one downside may be it is very resource intensive, and despite its thoroughness, it is still prone to mistakes. This is mostly because it is more measuring the images response to filters, vs directly categorizing them.

5 Texture Matching using Gabor Texture Features

Methods/Results

Similar to the previous section, Gabor textures can be used to help extrapolate texture features from images. The process works in a similar way where a target image is run through a set of filters, creating a n-dimensional texture feature based on the averaged filter response of each filter. Gabor filters are unique in that they often follow certain Gabor equations, which can make them larger or smaller, change their orientation, and their intensity. This means that certain features can be finely extracted from images using these filters. For this project, it was requested to use anywhere form 7-28 filters. the value of 12 was chosen, with filters ranging in shapes, sizes, and orientations. The figures below shows the 12 filters that were used.

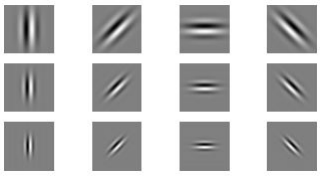

Figure 20: Gabor Filter Bank

It should be noted that the project allowed for the use of a 3rd party to generate the filters. The python code written for the project utilized other python code that was found online and has been referenced both within the code itself and in the reference section. This code was responsible for generating the bank of Gabor filters. Once the Gabor filters had been created, the same process as in the previous section was used to compare images against the data base. The Chi-Square distance was also used as well. The tables below show the results from 2 different sets of test with a target from each class next to the classes of the top 4 matching images.

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Bark
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Plaid	Wood	Wall	Wall
Carpet	Carpet	Carpet	Carpet	Carpet
Plaid	Plaid	Pebbles	Plaid	Plaid

Table 9: Gabor Results 1

After the results had been compiled, the 2 best cases and the worst case was

Query	Match1	Match2	Match3	Match4
Bark	Bark	Bark	Bark	Bark
Wood	Wood	Wood	Wood	Wood
Pebbles	Pebbles	Pebbles	Pebbles	Pebbles
Wall	Wall	Wall	Wall	Wall
Carpet	Carpet	Carpet	Carpet	Pebbles
Plaid	Plaid	Pebbles	Plaid	Plaid

Table 10: Gabor Results 2

selected. The results of which, along with the images themselves are displayed below.

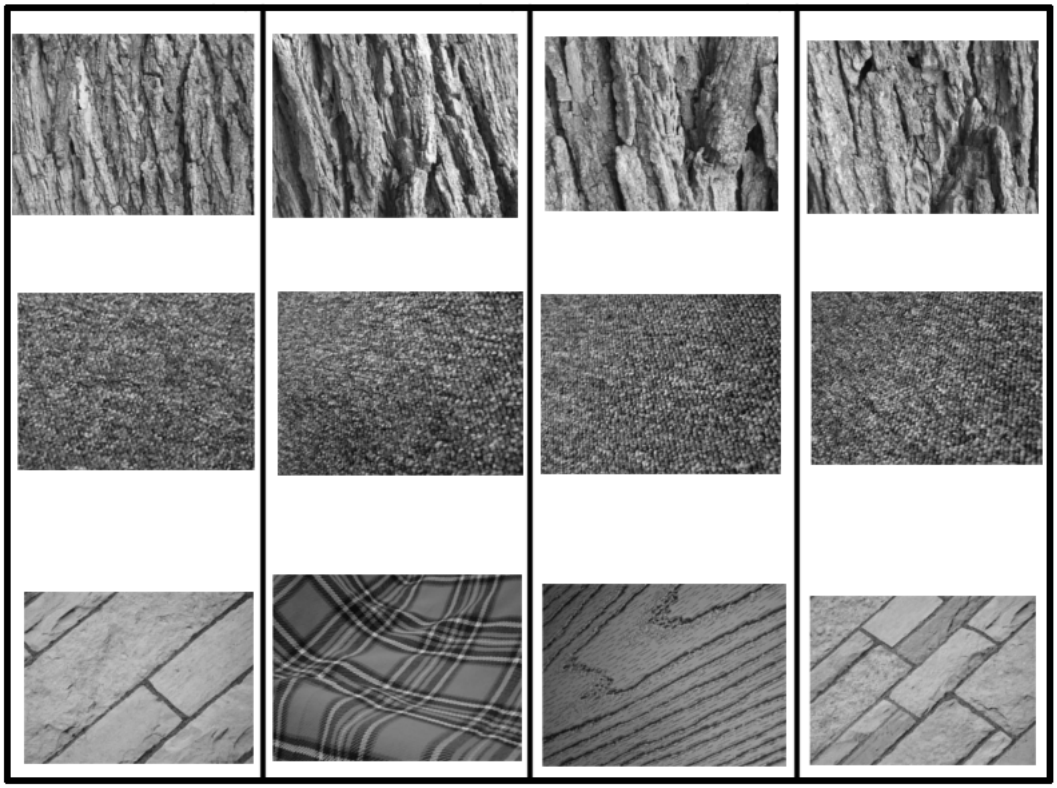

Figure 21: Gabor Best and Worst

As the results show the Gabor filters did a phenomenal job at getting accurate results. In fact, there were only 3 instances in total where the filters guessed the wrong texture. This speaks to the robustness of the Gabor filters

and how the use of a larger bank with more filters produces better results than seen in the previous section that had only 7 filters.

6 Discussion

Overall, the Project was able to achieve what was required and explored many different applications of image manipulations. The project also doubled in helping understand ways of how to implement these features in working code that could input and output the required results. Not only that, but by being able to create these features in a coding environment, it allowed for a more thorough understanding of the concepts by being able to get detailed information about what each step was doing. Lastly, it should be noted that the code was written and ran using Jupyter Notebook on a local machine. Overall, a detailed implementation of histogram intersections, texture filtering, and Gabor bank filtering was achieved and most of the results were expected.

References

Gabor filter bank: http://vision.psych.umn.edu/users/kersten/kersten-lab/courses /Psy5036W2017/Lectures/17$_{P}ythonForVision/Demos/html/2b.Gabor.html$