CS6640 A3
Cade Parkison
10/14/2018

Assignment 3: Frequency Domain Filtering

Note: Included in the assignment folder are a few files to help test my results. The main test file is "testing.m", which has separate code blocks for each problem that will show the results I obtained. Another testing file is titled "A3_testing.ipynb" which is a jupyter notebook I used in the development of my work. An easily viewable form of this file is titled "A3_testing.pdf", which is the pdf exported from the jupyter notebook. In order to run the .ipynb file, a matlab kernel is required for jupyter.

Problem 1: FFT Texture Feature Analysis

In this problem, the goal was to analyze texture features of images using the Fast Fourier Transform. In order to do this, I first looped over each pixel and found the Fourier Transform of the surrounding 5x5 window. To make life easier, the loop over the pixel excluded a 2-pixel border surrounding the entire image. This allowed my 5x5 windows to all contain the same number of pixels without having to zero pad the image. This had the effect of losing some data on the image extremities, however on the image and window scales used, this was hardly noticeable. Next, the power spectrum was calculated from each Fourier Transformed window. This power spectrum was then used to populate a texture feature array over the entire image.

Below, in figure 1, is the sample image used for most of my testing.

Figure 1: Original Test Image

After the texture feature array was made, this was passed along to Matlab's k-means algorithm in order to partition the pixels into clusters based on texture similarity. In testing, I tried various k-values as the second argument in the kmeans() function. K-values of 5,7, and 9 were tested, but in the end, the best results were obtained using a k-value of 7. This gave enough distinctive pixel regions to extract meaningful data about the underlying image textures.

Figure 2 below shows the results of this k-means clustering. As can be seen in the K=2 figure, this method does a good job of segmenting the areas of uniform texture like the sky and the light colored concrete. An interesting phenomenon occurred when plotting and exporting the figure below. It seems Matlab removed some of the top and right border pixels when I copied over the figure with the subplots below. This can be seen best in the K=5 figure, where the red border can only be seen on the bottom and right sides of the image. This was not present in the original plots generated by Matlab. This K=5 figure also shows the effects of not using any zero padding, where the red lines represent the 2-pixel boundary caused from generating the 5x5 pixel windows.

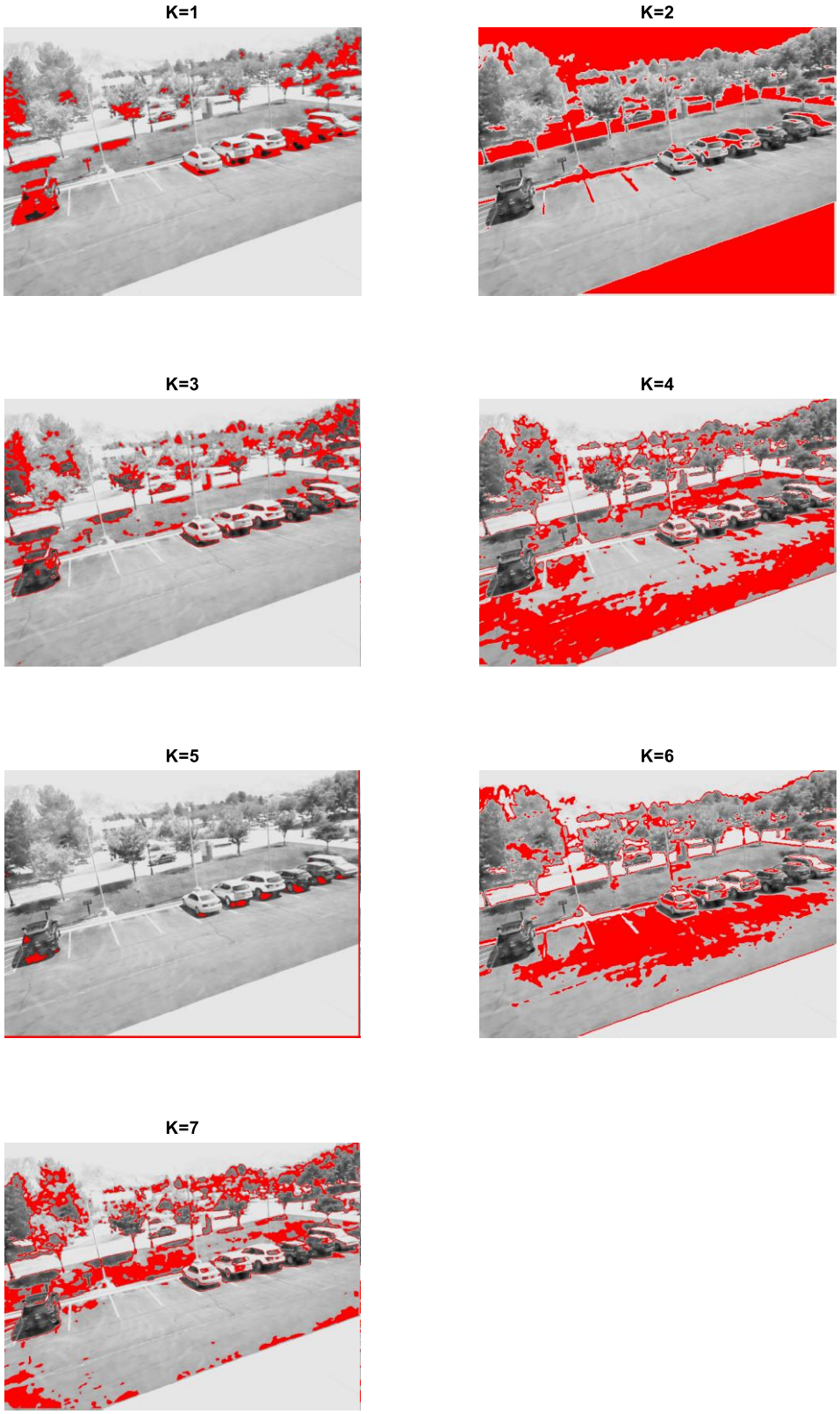


Figure 2: K-means results for Problem 1

As in the last assignment, I think this method of texture feature analysis is promising, even more so when used with pre-processing such as some form of edge detection.

Problem 2: FFT Radial Texture Feature Analysis

This problem was similar in structure to the previous one, but this time a different metric was used to generate the texture parameters. Instead of taking each pixel of the power spectrum as a separate feature, concentric radial segments were taken from the power spectrum and each of these segments were integrated to generate one texture parameter. To have 10 radial segments of 1-pixel width, a window of 19x19 was used instead of 5x5.

To segment the power spectrum into annular components, I created a function titled "CS6640_rings.m". This function generates a cell array of length 10, where each element is an Nx1 matrix containing N linear indices corresponding to the (x,y) indices of the 19x19 window. This was done by looping over each of the 19x19 pixels and calculating the L-10 norm from its location to the center of the window. The norm is then used to separate each pixel into its corresponding segment. Next, this cell array of indices is used in the CS6640_FFT_radial function to index each segment of the power spectrum which are then integrated using a sum of squares of each element to form the texture parameters.

As in the previous problem, these texture parameters are fed into a k-means clustering algorithm to extract meaningful data about the underlying image textures. These results are shown below in figure 3. A few observations are of interest. First, due to the larger size of the window, 19x19 instead of 5x5, this method revealed lower frequency information. This is visible in the smoother texture regions in the clustering as compared to Problem 1. An effect of using annular components was that the texture elements were more "blob-like". This is most present in the K=4 image below, where "blob-like" clusters of pixels are seen in the trees. Again, the effect of not using zero padding is evident in the K=3 image, this time much more apparent. A 9-pixel border can be seen in most of the images, where information is lost due to the way the window was moved across the image. Even if zero padding is used, it is not possible to extract very meaningful data about these edge pixels.

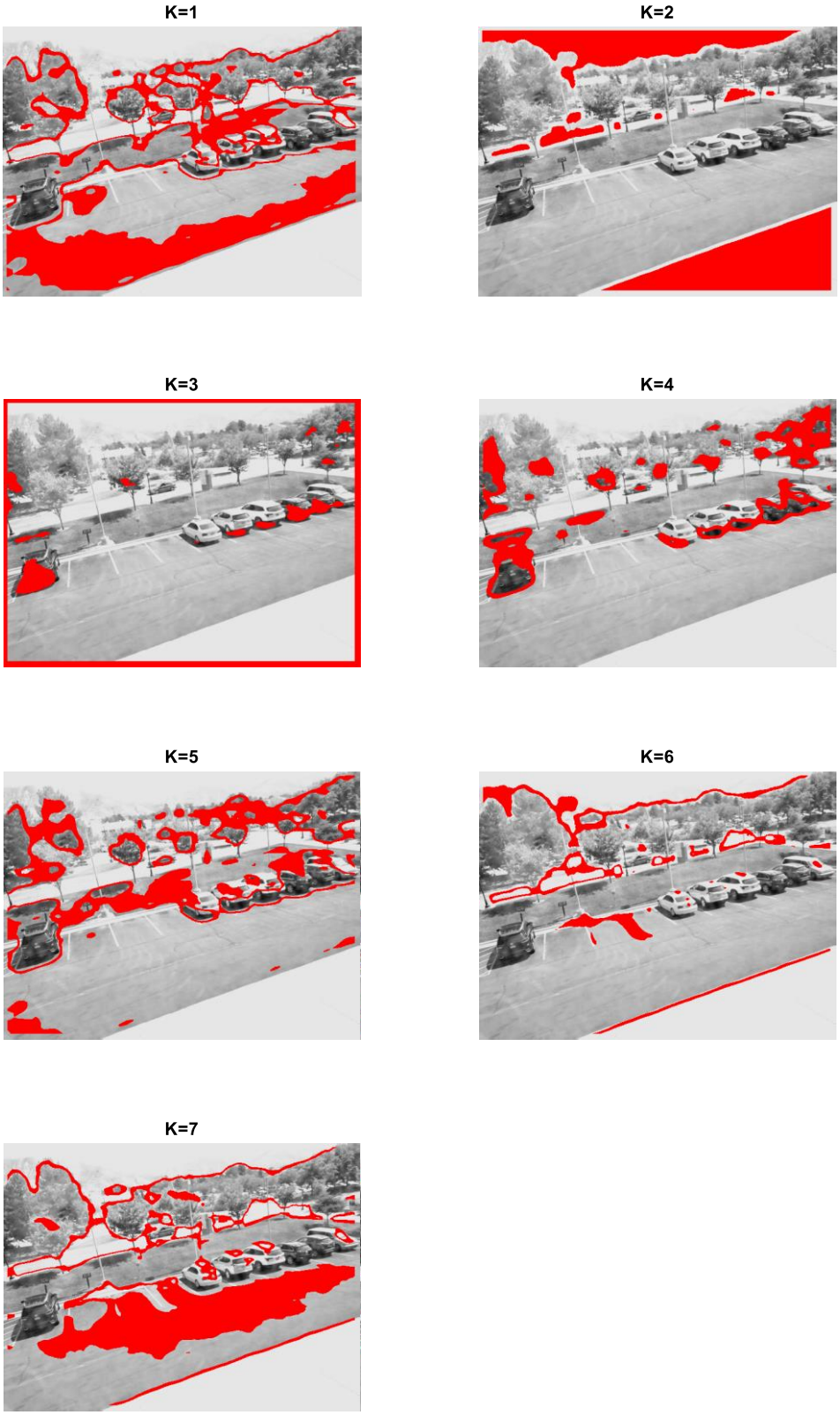


Figure 3: K-means results for problem 2

Problem 3: FFT Angular Texture Feature Analysis

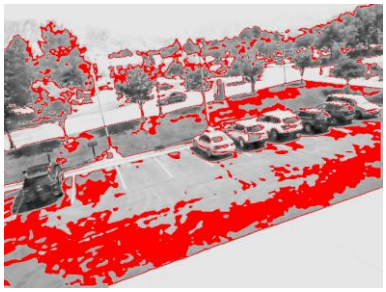
In problem 3, the same idea was explored but this time using another metric for the texture parameter separation. Angular segments of the power spectrum were used to do this. I wrote a function called CS6640_angular(w), that given a window size w, will return the pixel linear indices of 8 angular regions in that window. These 8 angular regions are each 45 degrees in angle.

As in the previous problem, the power spectrum from each (w x w) window was segmented acording to the previously found indices. Each segment was then integrated into one number by taking the sum squares of all the power spectrum values. After this is computed for all pixels in the image, the texture parameters are then passed along to k-means for clustering. The same k-value of 7 was used in this problem to make it easier to compare the results to the previous problems.

For an interesting comparison, I decided to test the effects of window size on the texture clusters. Window sizes of 5, 11, and 19 were used. As the window size increases more high-frequency data is lost, this is visualized by the increase in smoothness of the texture regions from 5 to 11 to 19 sized windows.

Overall, the angular texture analysis method seemed to be very promising in separating regions of similar directionality. These wedges seem to almost find the edges in the image. This is best seen in Figure 4 below, particularly the K=2 and K=3 image. In the K=2 image, the road on which the car is moving is mostly classified into one region, with disctinct lines formed where the tree trunks are in front of the road. In the K=3 image below, the dark regions around the cars are segmented together nicely, distinctly showing the edges of the cars shadows.

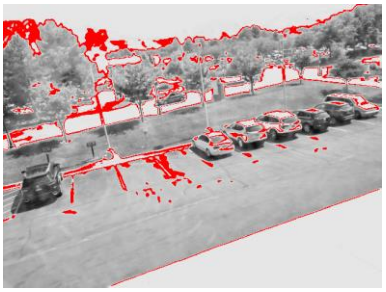
K=1



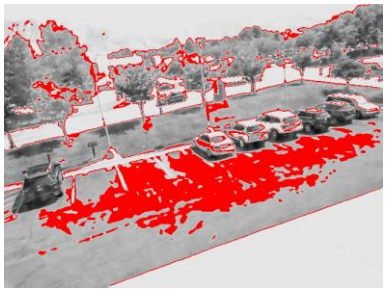
K=2

K=3

K=4



K=5

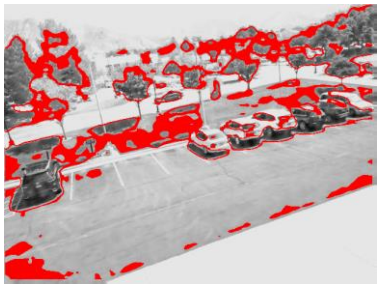


K=6

K=7

Figure 4: K-means with window size of 5

K=1

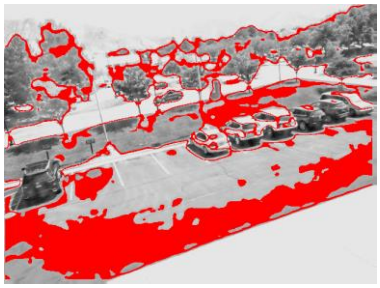


K=2

K=3

K=4

K=5



K=6

K=7

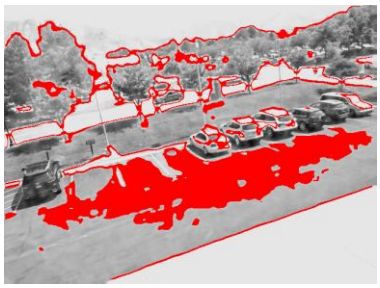


Figure 5: K-means with window size of 11

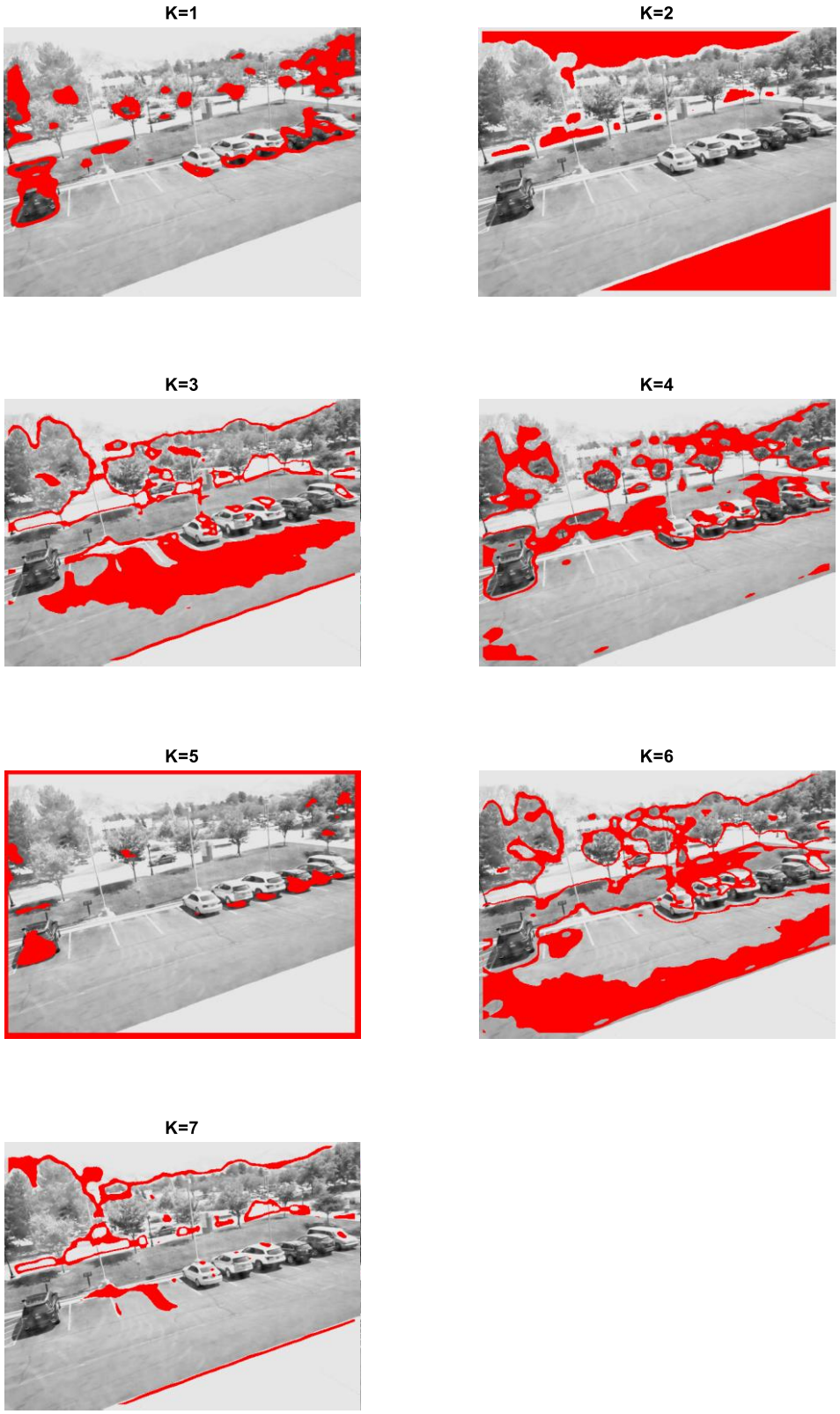


Figure 6: k-means with window size of 19

Problem 4: Fourier Shape Descriptors

This last problem proved to be very tricky for me with the provided information. After much thought, I believe I have the function working as intended. After inputting the closed curve as well as the distance between points, my function calculated the absolute angle between each pair of adjacent points. Using this angle as well as the starting angle, a "cumulative angular bend" is calculated for each point and stored as a vector. This vector measures the way in which the shape differs from a circular shape. A vector with a large norm would represent a shape that is much more "stretched out" and jagged compared to a circle. In this sense, a shape "H" should have a larger norm compared to that of a shape "O".

Below are the two input shapes used for testing of the function. As can be seen on the left, I had to remove a point from the top left of the H to get an even number of points. This was to make extracting the shape descriptors easier. I do not think this simplification resulted in much of an error in the results.

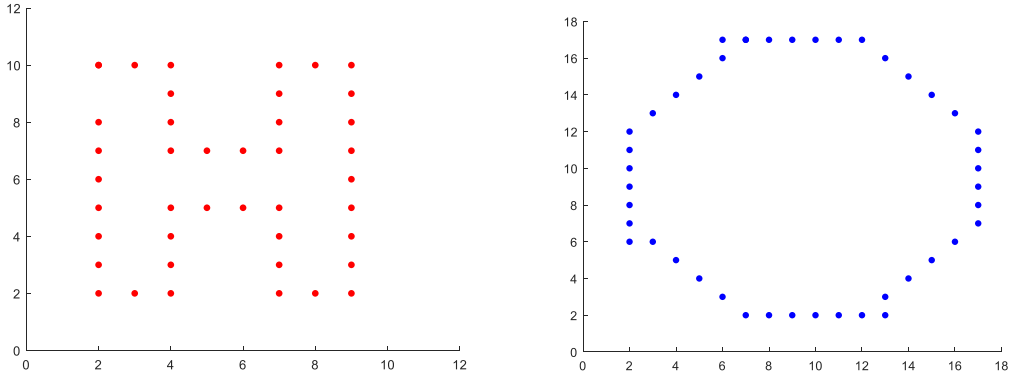


Figure 7: Input shapes

The output of my CS6640_FFT_shape function is a vector of shape descriptors, which are complex numbers resulting from the FFT function. To better visualize this data, I took the magnitudes of these complex number vectors and displayed them as a bar graph below. The X axis represents the different frequencies, and the Y axis the magnitudes of those frequencies.

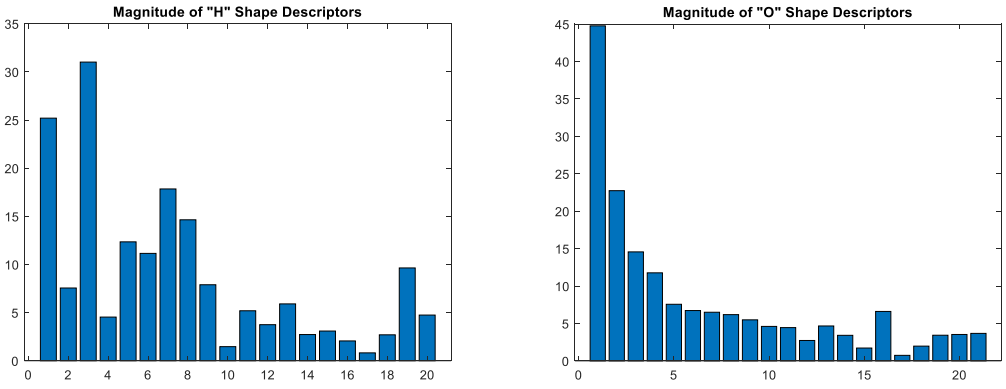


Figure 8: Magnitudes of Fourier Descriptors

A few remarks can be made from the images of the shape descriptors above. First, the “H” seems to have higher magnitudes at the middle and high frequencies. I believe this is due to the “H” having more sharp corners, and overall less like a circle, as compared to the “O”. The “O” shape tends to have it’s highest magnitudes at the lower frequencies. I believe this data could be improved if it were not for the unusual vertices in the “O” shape, where these 90 degree corners are present.

Overall, this experiment was valuable in learning more about the FFT and the usefulness of its results. I think this approach could be applied in a neural net to classify shapes according to their shape descriptors. A large test sample of ground truth data could be used to train the neural net to classify new shapes in images. For this to work well, some pre-processing would need to be done on the images, such as filtering and edge detection to generate the outer boundary of the shapes. Care would need to be taken so that the curves of the boundaries were closed as well.