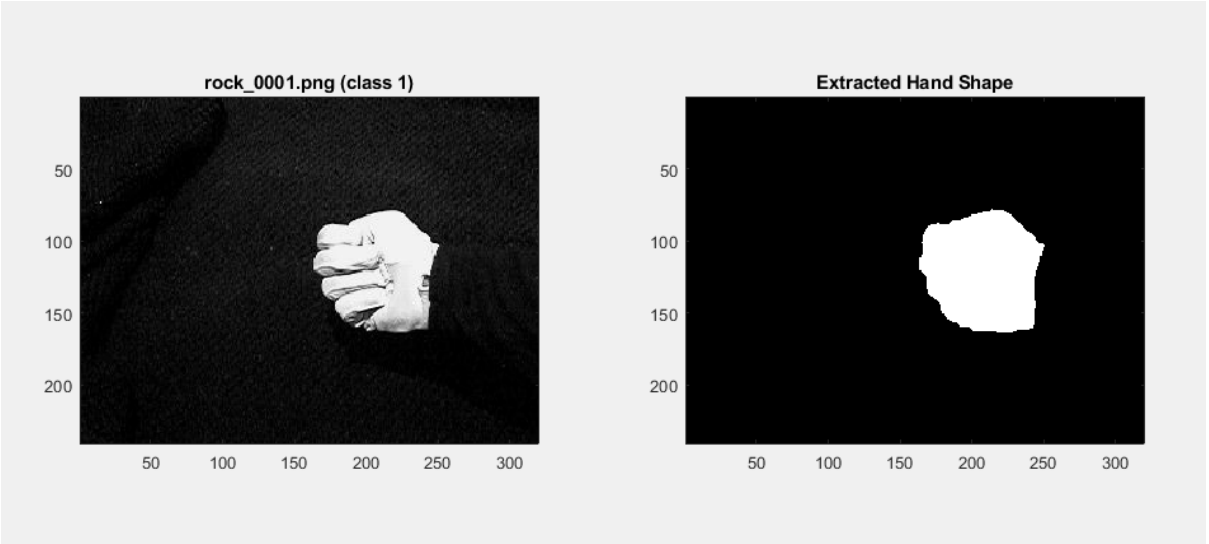
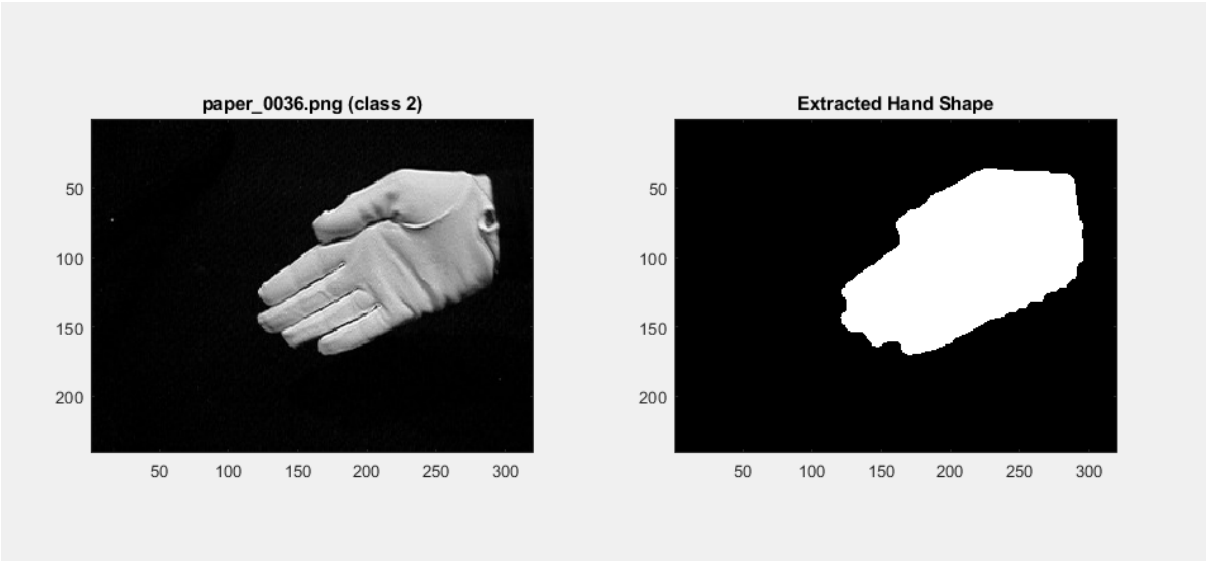
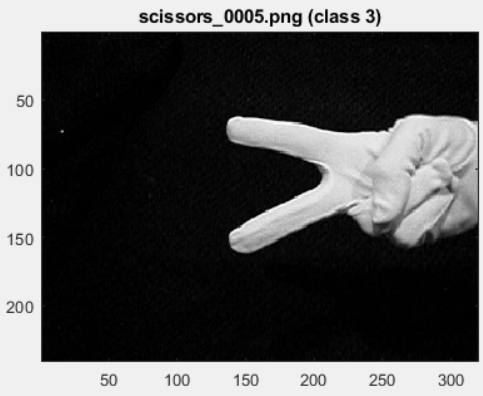
Assignment 4

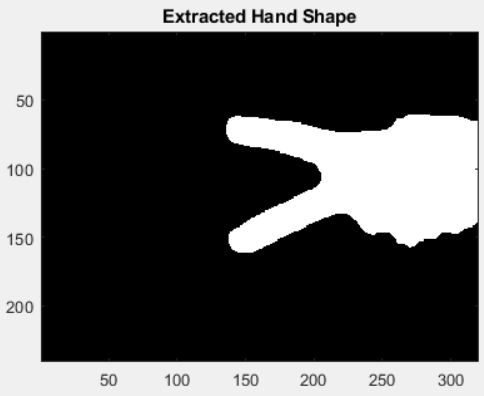
Exercise 4A

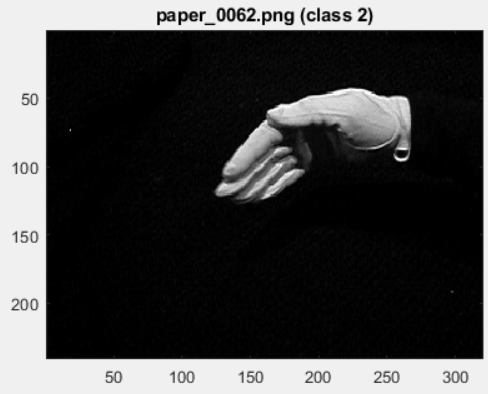
Results (0.45 Scaled Threshold)

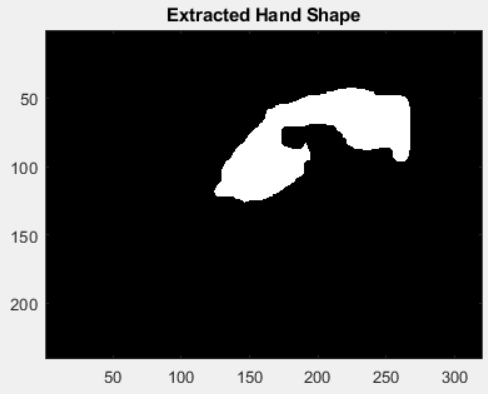


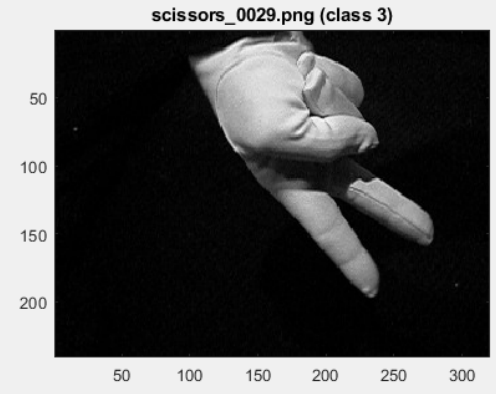


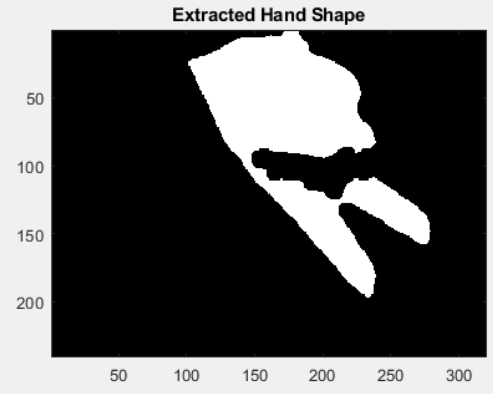

scissors_0005.png (class 3)
50
100
150
200
50 100 150 200 250 300


Extracted Hand Shape
50
100
150
200
50 100 150 200 250 300

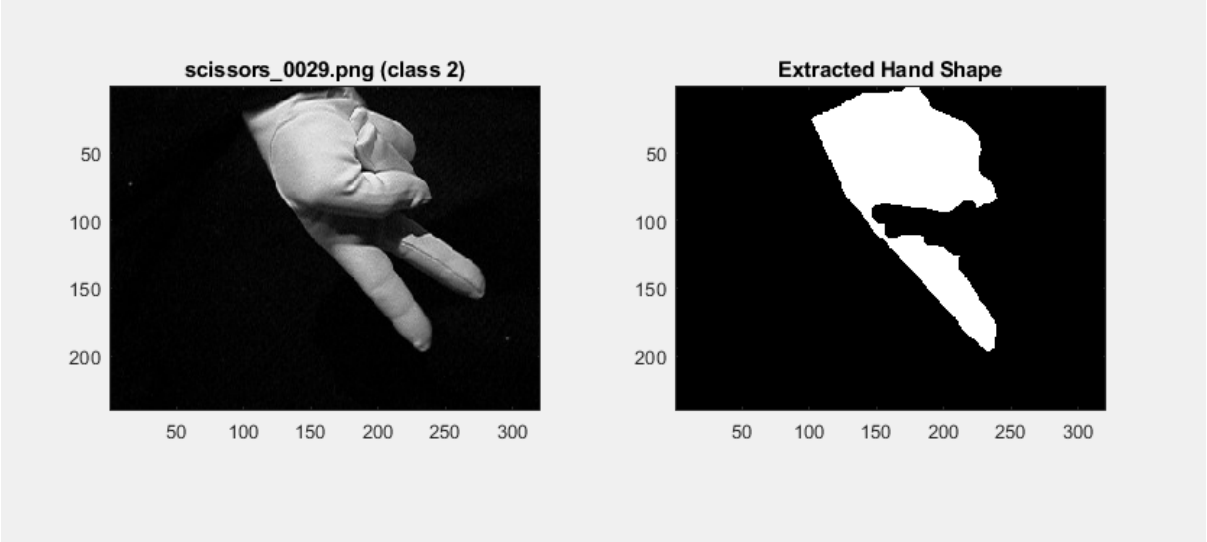

paper_0062.png (class 2)
50
100
150
200
50 100 150 200 250 300


Extracted Hand Shape
50
100
150
200
50 100 150 200 250 300

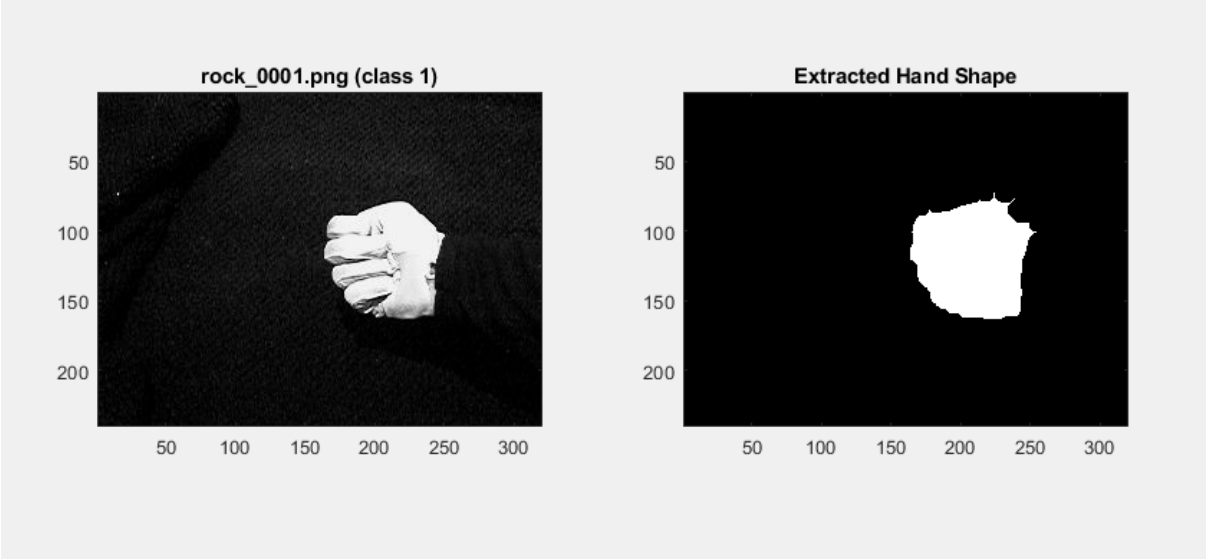

scissors_0029.png (class 3)
50
100
150
200
50 100 150 200 250 300


Extracted Hand Shape
50
100
150
200
50 100 150 200 250 300

Results (0.5 Scaled Threshold)



Results (0.4 Scaled Threshold)



Comments

The `hand_threshold` and `hand_extract` function was able to successfully extract the hand from the original image. The resulting image was a 2 bit image that contained only a single connected region. The process used to create the thresholding was exactly the same process given in the lecture. This was done 20 times. The resulting threshold was then scaled to 0.45. This scale was chosen as this was the most appropriate middle ground between 0.4, which had the complete shape of the hands in all of the image but had a lot of noise, and 0.5 which had most of the shape and resulted in no noise.

The scaling is important to get right at this point since this would greatly affect the final results of the classification of the images. For example, with a 0.5 scaled threshold, `scissors_0029.png` would lose a finger, while at 0.45 this finger was retained. On the other side of the scale, `rock_0001.png` has some visible noise at the edge of the thumb when the threshold was scaled at 0.4. This was not present when the threshold was scaled at 0.45.

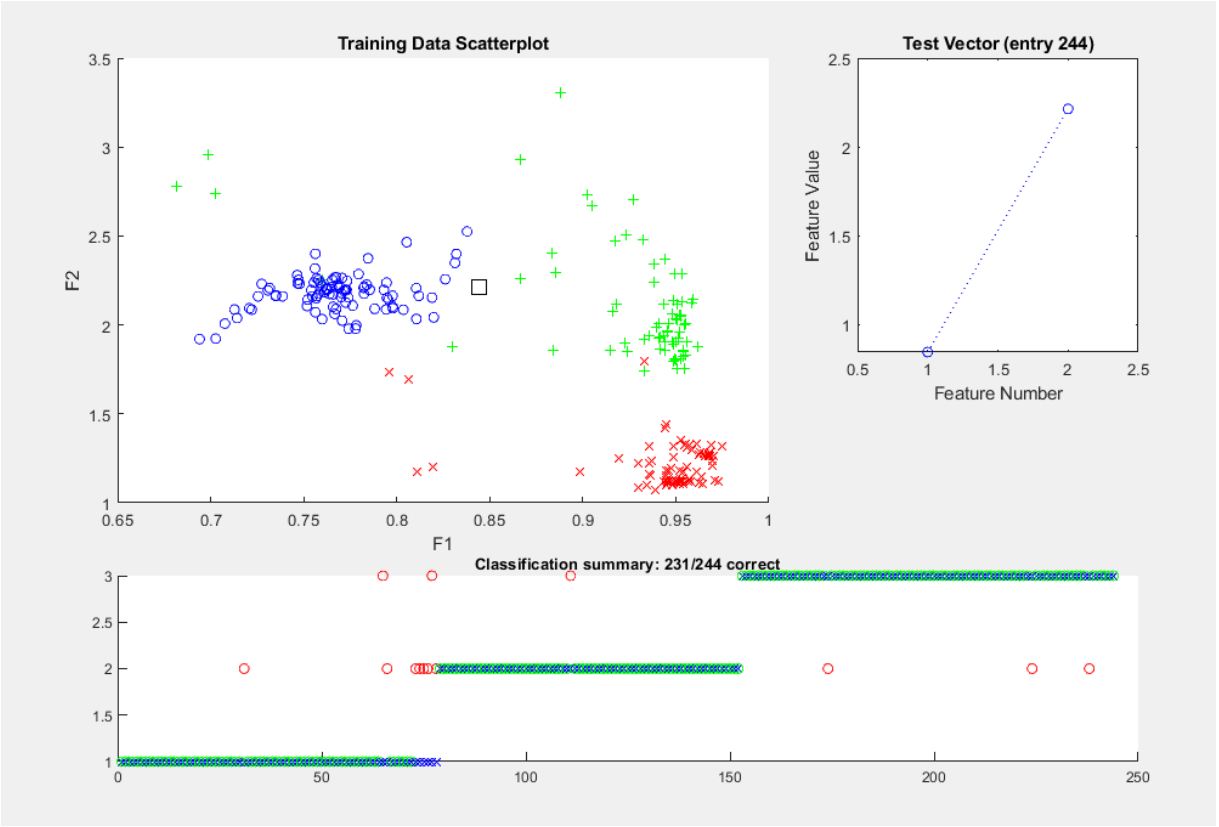
However, even with 0.45 scaling, some images, like `paper_0062.png` and `scissors_0029.png` still had visible gaps within the shape of the hand due to shadows.

The resulting image was then cleaned using morphological filters. The filters used were `imclose` and `imfill`. `imclose` was chosen to try to connect instances where outlines do not connect and to try to close small holes without drastically altering the shape of the object. The shape of the morphological structuring element was a disk with a radius of 6. `imfill` was done after closing to fill in any missing holes that weren't filled by `imclose`. A singular blob was then chosen through the use of bwlabel.

The total time to extract a single image was 0.422s. The line that took the most time, 64.3%, was the function that created the morphological structuring element. The next few lines were `imfill`, `hand_threshold`, and `imclose` with 18.7%, 6.2% and 4.9% of the total running time, respectively.

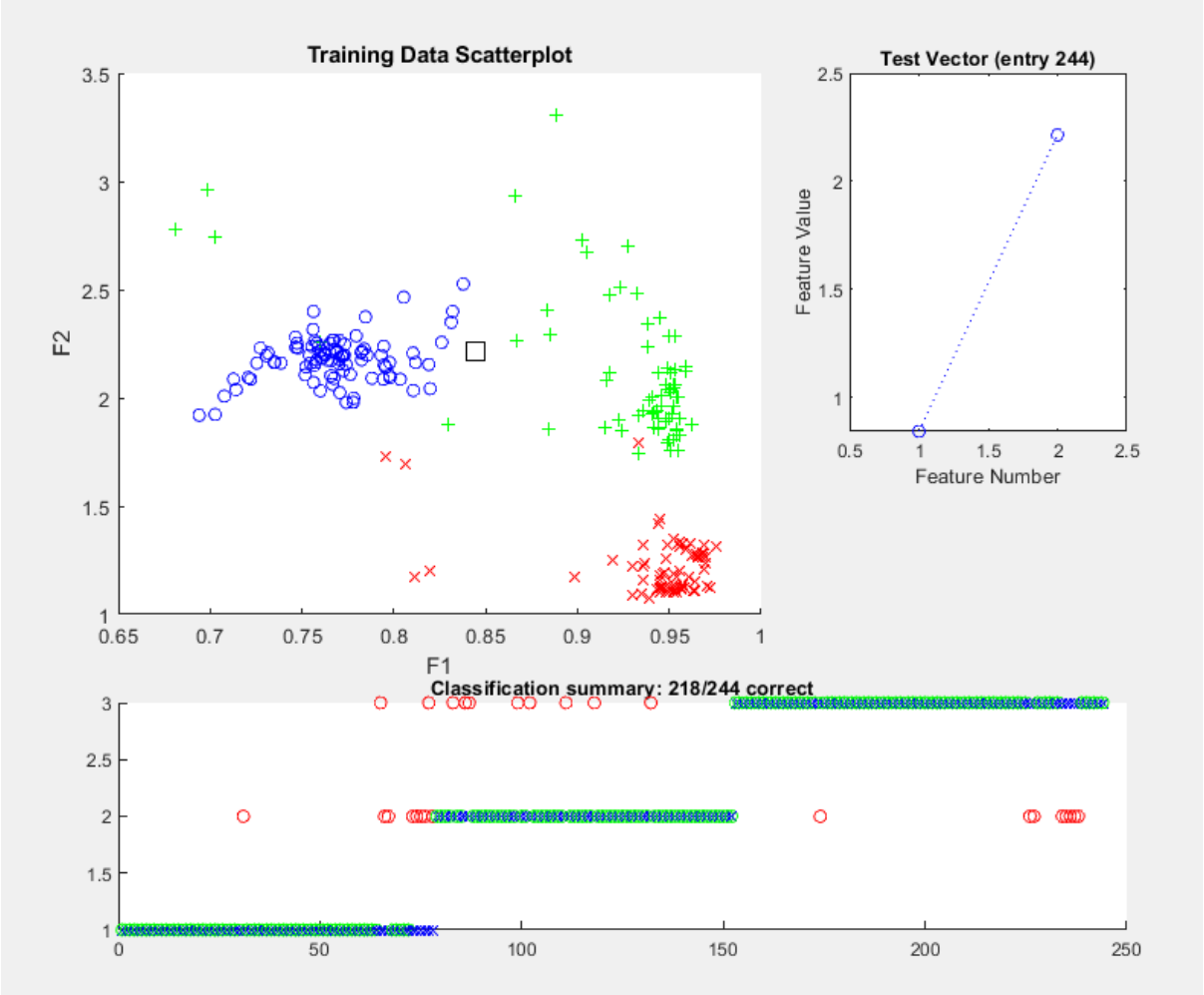
Exercise 4B

Results (Solidity & Aspect Ratio; k=3)

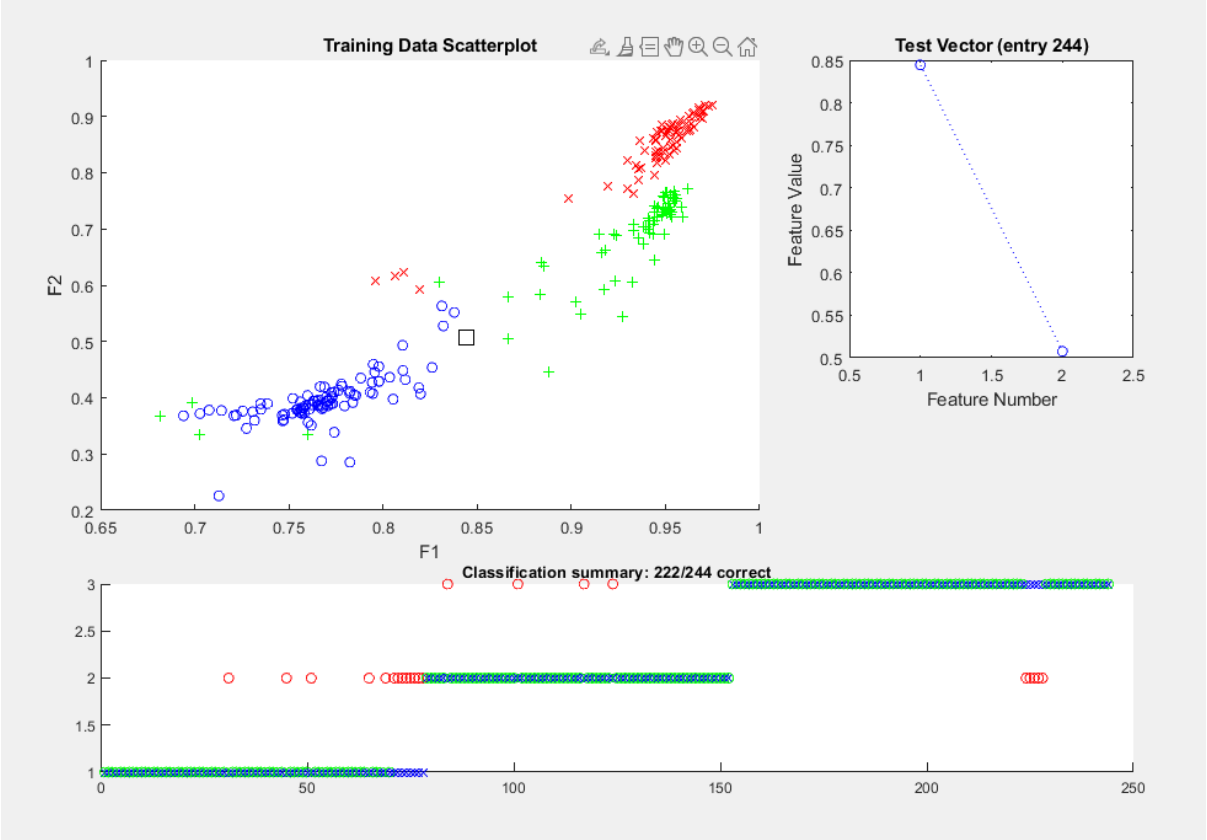


Results (Solidity & Aspect Ratio; k=15)

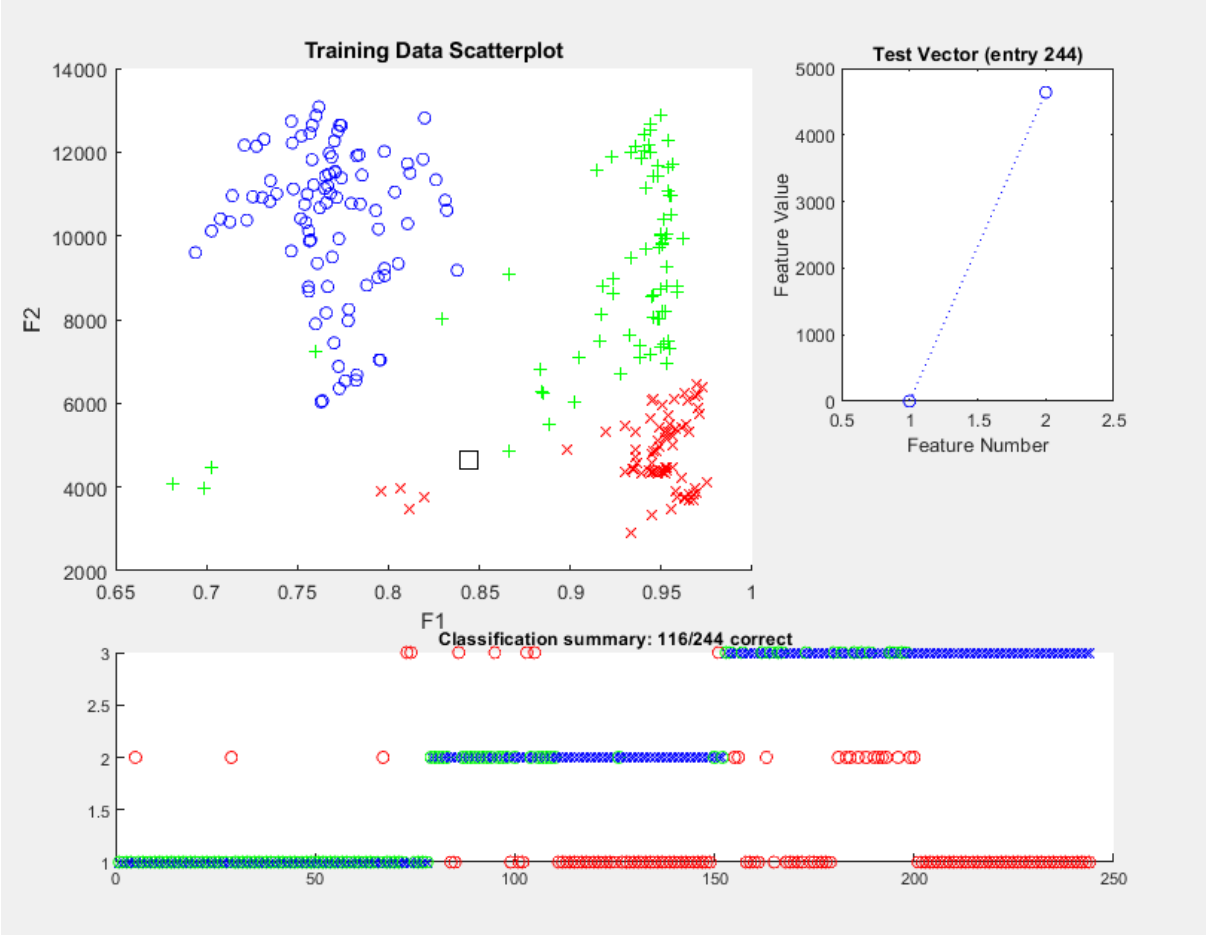
Results (Solidity & Aspect Ratio; k=30)



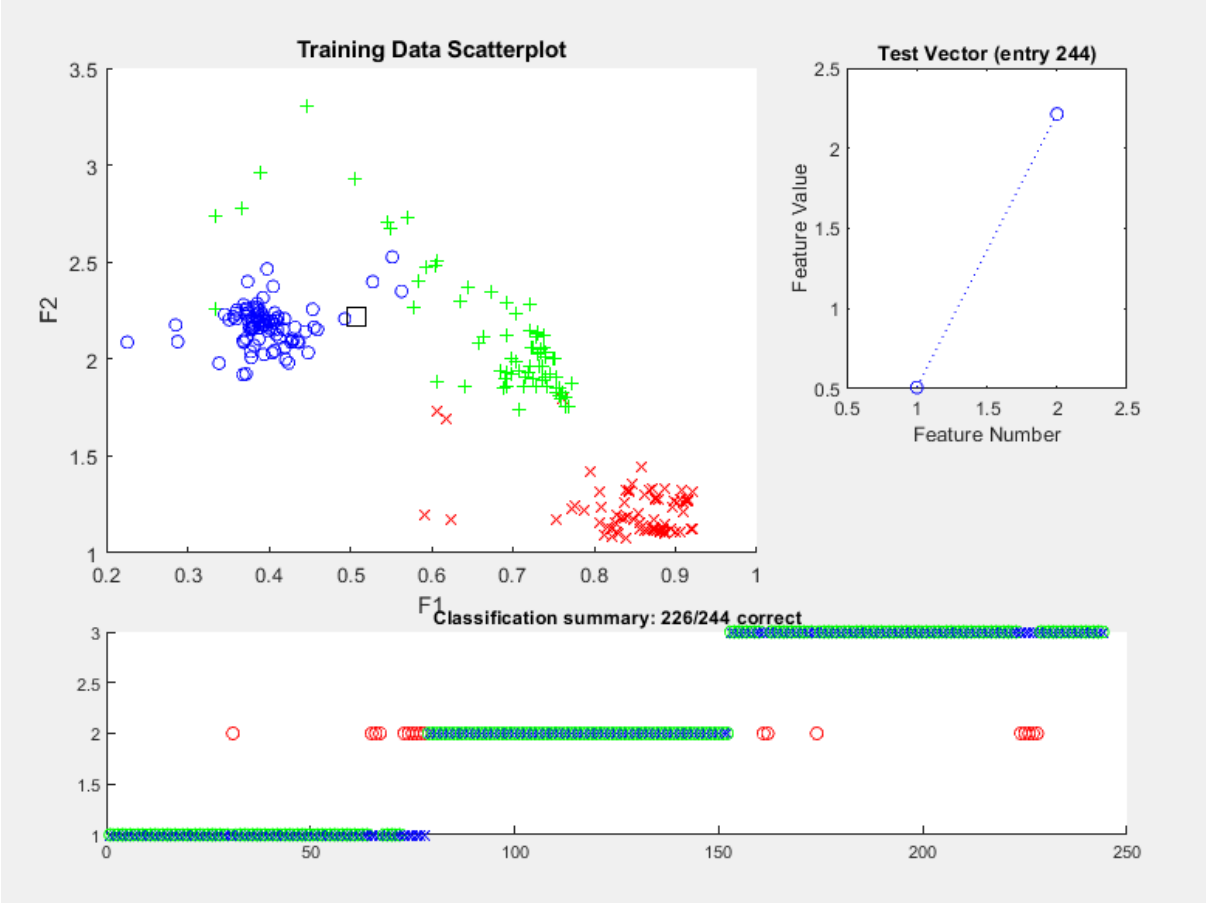
Results (Solidity & Circularity; k=3)



Results (Solidity & Roundness; k=3)



Results (Aspect Ratio & Circularity; k=3)



Comments

The `hand_features` function was able to successfully return two features that accurately classified the different hand shapes into rock, paper, or scissor. These features were generated using `regionprops`. These features were then used as input for the k-nearest neighbors algorithm that classified each image.

Choosing the set of features to use in classifying the images were the result of a trial and error process where different features that can be obtained from `regionprops` were tested and compared until a set of features obtained accuracy ratings above 85+%. These tests using KNN were done with k=3.

The first few features tested were features that involved getting the convex hull of the shape. The idea here is that each shape should have different convex hulls because their shape is different. For instance, the convex hull of a scissor image should contain more points compared to a convex hull of a rock. This was done by using the `'ConvexHull'` as the property to be returned by `regionprops` and counting the number of points. This proved to be fruitless as accuracy was less than 50%, no matter what the second feature was.

More metrics that corresponded to the convex hull of the images were tried as the idea seemed to make sense even if the results did not show great promise. As such, Solidity was the next choice. The test with circularity as the second feature showed great accuracy: 90.98%.

Since solidity gave great results when combined with circularity, I tried more tests with solidity as the main feature. I tried roundness. The formula used was:

$$\frac{\pi}{4(features.Area)}$$

This formula was acquired based on the following process:

Given:

$$\frac{4(area)}{\pi(max\_diameter)}$$

Substituting Max Diameter with formula from EquivDiameter:

$$\frac{4(area)}{\pi(\frac{4(area)}{\pi})^2}$$

Simplifying:

$$\frac{\pi}{4(area)}$$

Using the value returned from `regionprop`

$$\frac{\pi}{4(features.Area)}$$

This combination of solidity and roundness resulted in an accuracy of 47.54%. This combination was scrapped since the accuracy was well below 85%. Other tested feature sets also had accuracy in this range and were omitted for brevity as they are not that interesting.

The final chosen feature set was solidity and aspect ratio. Aspect ratio was obtained by dividing `'MajorAxisLength'` by `'MinorAxisLength'`, as given by the formula in the lecture. This was the combination that resulted with the highest accuracy so far: 94.67%.

Other features were also tested with aspect ratio. The only feature pair that was interesting was circularity as it had an accuracy of 92.62%.

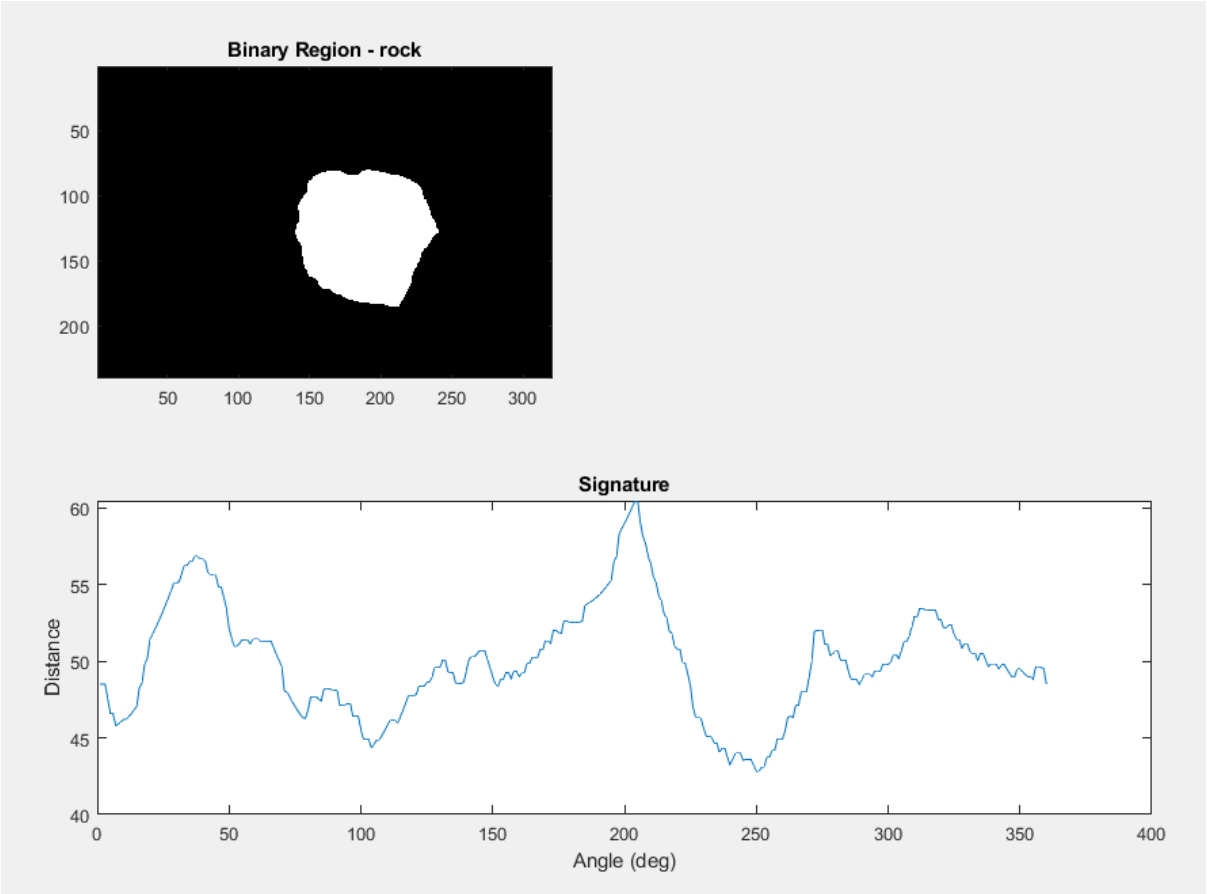
From these testing, it was determined that three features had great effect in differentiating the three hand shapes: Solidity, Aspect Ratio, and Circularity. From these three features, the combination that had the greatest accuracy was solidity and aspect ratio. This feature set was tested with a higher number of k in order to determine if having a higher number of neighbors would help this feature set. With k = 15 and k = 30, the accuracy of the classification continued to decrease as k increased, with accuracies of 93.03% and 89.34%, respectively.

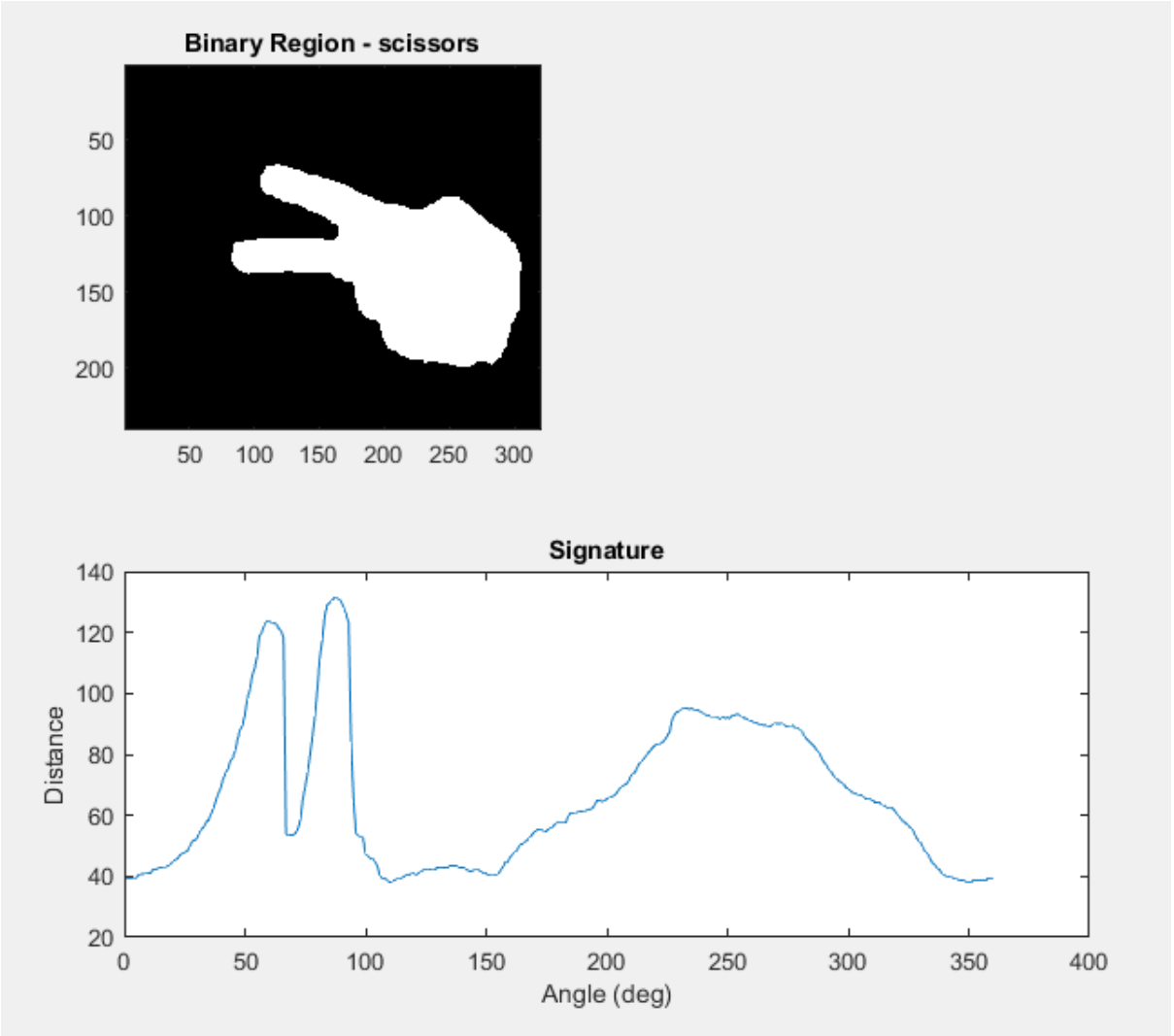
As such, for this data set, the recommended feature set is solidity and aspect ratio. In conjunction, for greater accuracy, k should be set at 3 for the KNN algorithm.

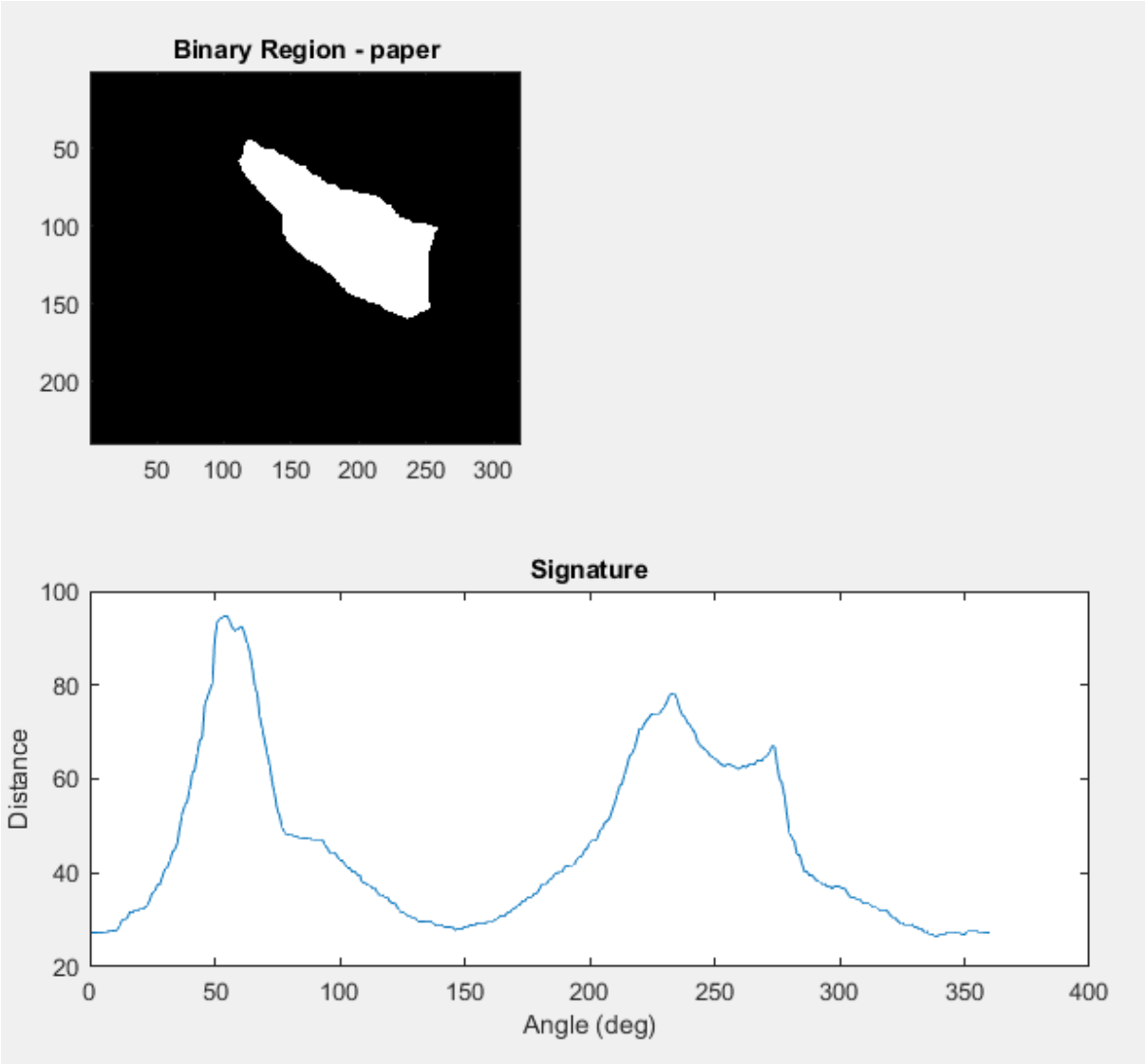
In terms of performance, The total time it took to run the test was 77.536s. The majority of the time was spent in `all_hand_features` with it taking 80.5% of the total time. However, the majority of the time was spent in drawing the figures. The time spent in extracting the hand and generating the features only accounted for 7.5% of the running time of the function. The same was also true for `all_hand_classify` which accounted for 18.3% of the running time of the test. `knn_classify` only took 0.1% of the running time of the function. The rest were spent in generating the figures.

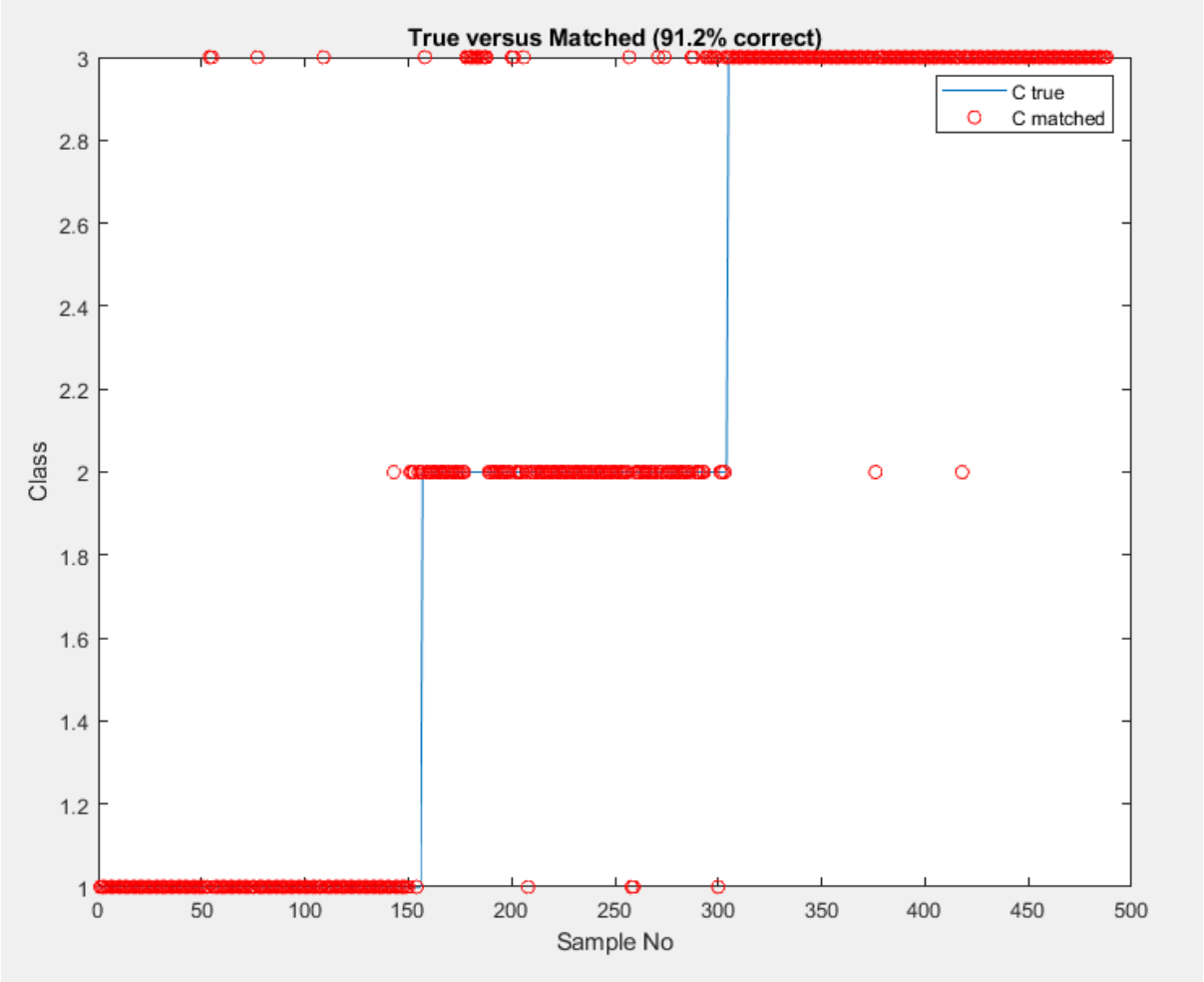
Exercise 4C

Results




Binary Region - scissors
50
100
150
200
50 100 150 200 250 300
Signature
Distance
140
120
100
80
60
40
20
0 50 100 150 200 250 300 350 400
Angle (deg)


Binary Region - paper
50
100
150
200
50 100 150 200 250 300
Signature
Distance
100
80
60
40
20
0 50 100 150 200 250 300 350 400
Angle (deg)


True versus Matched (91.2% correct)
C true
C matched
Class
Sample No
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
50
100
150
200
250
300
350
400
450
500

Comments

The function `hand_signature` was able to successfully get the boundary signature of the 3 different shapes when run using signature_test. The total time to run signature_test was 0.887s, with `hand_extract` taking 13.6% of the time and hand_signature taking 0.121s. The line that took the most time was `bwboundaries` and `regionprops`, with both taking 0.029s or 24.1% of the total running time each. The next line that took the most time was the line that checked if the current y' or the rotated y is within two pixels of the y axis and determining if that y' is greater than the current maximum.

Signature classification through `signature_classify` resulted in an accuracy of 91.2%. The total time to run this classification is 70.116s. Just like in feature classification, the majority of the time was spent in generating the figures. Extracting the hand accounted for 6.5% of the total running time while creating the hand signatures took 13.7%.

This type of classification had less accuracy compared to feature classification by 1.55%. This means that the two classifications are neck and neck in accuracy. It should be noted that signature classification is faster than feature classification by 7.42s when run on the same data set. This means that choosing which process to be used in classification depends on the use case. If accuracy is more important, then feature classification is to be used, knowing that there is a tradeoff in time. If speed is of greater importance, then signature classification should be used, knowing that there is a very slight decrease in accuracy. However, this generalization may not be true for all data sets. This is only true for this specific images.

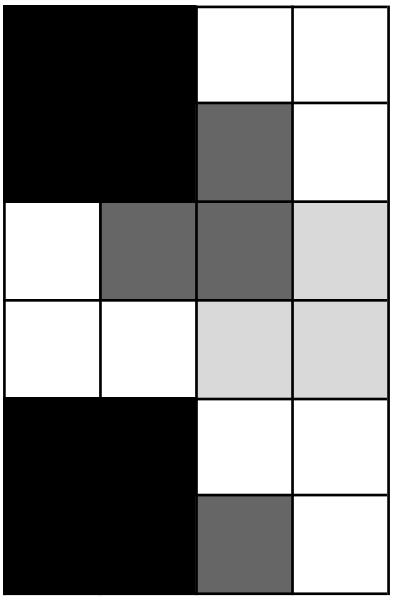
Exercise 4D

Question 1

I realized that I can simplify the problem. Instead of manually counting all 64 pixels in the texture, I can count only 16 pixels, reducing the problem area to ¼ of the original size. This was because the first 4x4 area is simply repeated 3 more times for a total of 4 4x4 areas that are exactly the same. This was basically having a 4x4 area with the texture repeating at the boundaries. With this in mind, I was able to come up with a process for both a and b.

A: 2 Pixels Down

I took the 4x4 area and added 2 additional rows at the bottom, simply repeating the texture:



From there, I manually counted the pixel pairs in the 4x4 area and tabulated them

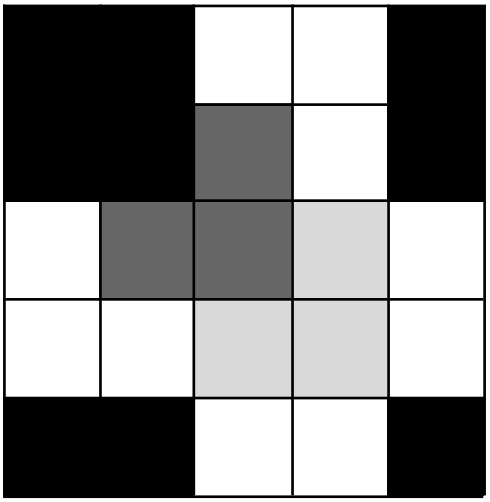
(i,j)	0	1	2	3
0	0	1	0	3
1	1	0	1	1
2	0	1	0	2
3	3	1	2	0

After I have manually counted each pixel, I multiplied all the cells by 4, to get the co-occurrence matrix for the entire texture.

(i,j)	0	1	2	3
0	0	4	0	12
1	4	0	4	4
2	0	4	0	8
3	12	4	8	0

B: 1 Pixel Down and to the Right

I took the 4x4 area and added 1 additional row at the bottom and 1 additional row to the right, simply repeating the texture:



From there, I manually counted the pixel pairs in the 4x4 area and tabulated them

(i,j)	0	1	2	3
0	1	3	0	0
1	0	0	3	0
2	1	0	0	2
3	2	0	0	4

After I have manually counted each pixel, I multiplied all the cells by 4, to get the co-occurrence matrix for the entire texture.

(i,j)	0	1	2	3
0	4	12	0	0
1	0	0	12	0
2	4	0	0	8
3	8	0	0	16

Question 2

This algorithm was described briefly in lecture 10. Thickness was defined as the mean distance to the skeleton.

From this an algorithm can be constructed:

1. Perform a skeletonization process of the object using a hit or miss morphological filter.
2. For all pixels in the image, calculate the euclidean distance to the nearest skeleton.
    a. This can be done by iterating through all of the skeleton pixels, and finding the euclidean distance of each. The minimum euclidean distance from the previous process is then stored.
    b. This can also be done by doing a search around a radius. A BFS may be the most apt algorithm to find the nearest skeleton.
3. Average all the distances to get the final thickness value.
4. Multiply the final thickness value by 2.

Another way of naively solving this problem is by simply counting all the pixels in the image. However, this algorithm breaks down when an object is long but thin, and another object is short but thick. The thin object may contain more pixels than the thick object due to its length. The algorithm presented above does not have this problem. By calculating the distance of a pixel to the nearest skeleton, this removes the effect of length and the number of pixels on the calculation, focusing entirely on the width of an object at a specific point. The final thickness value is multiplied by 2. This is because the average distance is only half of the total width as the distance is not calculated from one end to the other, but rather from one end to the center.

After it has finished, this algorithm gets the normalized value of the thickness of the shape. A higher final value means a thicker object.