COMP422

Project 1: Data Mining Images

William Wallace
300411313
22/08/2020

PREPROCESSING

1.1 Edge Detection

For this edge detection algorithm, I started by importing the numpy library for scientific computing, the imageio library for image handling, and the imread module from the matplotlib library. The first method names two variables according the input image's height and width, and initialises a matrix with the same size, containing zeros for each element. In the same method, the values for each element in the new image are calculated to be the sum of the elements in the corresponding 3x3 matrix of the original image. This is performed in an element-wise fashion.

In the second method, two 3x3 kernels are initialised, the first kernel is a row mask with the values [-1, 0, 1], [-2, 0, 2], [1, 0, 1], while the column mask has the values [-1, -2, -1], [0, 0, 0], [1, 2, 1]. The row mask detects horizontal changes in brightness, while the column mask detects vertical changes. The Sobel operation is applied by calling the prior method to convolve the masks with the original image to calculate approximations of the gradient of the image intensity function. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using $\sqrt{(Gx^2+Gy^2)}$, where Gx and Gy are two images which at each point contain the vertical and horizontal derivative approximation. From here, the gradient's direction can also be calculated.

Finally, the main method is used to put the methods into action. It converts the input image into an array of values, applies the Sobel filter, and thresholds the image to make it black and white to better display the edges when the new image is seen. Finally, the image array is converted to 8-bit unsigned integer so minimise loss of information, before being written into a JPG format.

The edges of the input image are detected and highlighted in the output image shown in Figure 1. I noticed that there are some unexplained artefacts, for example in the upper right corner of the small boxes at the top left of the image. The Sobel edge detection algorithm works by surrounding the contrasting parts of the image by lines, which explains why small dots of images appear the have a thick border. When images are noisy as with

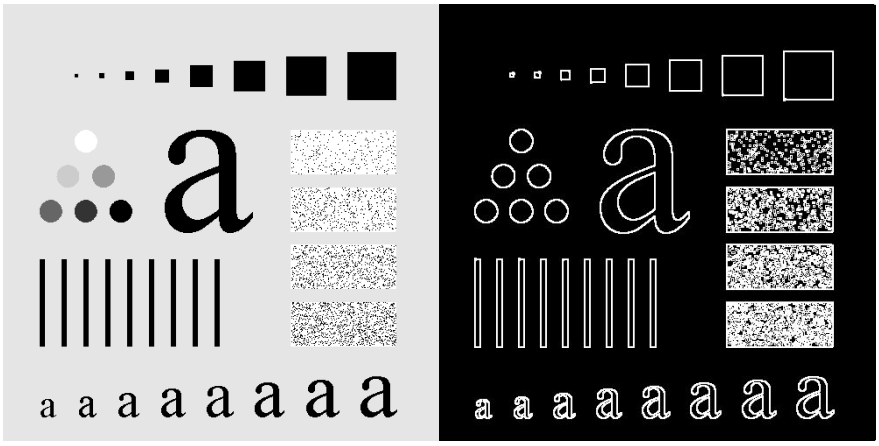


Figure 1. Initial image(left) and final image after Sobel edge detection with thresholding (right).

the boxes containing dots, the Sobel algorithm appears to struggle. This is likely because it is difficult to find a threshold for which noise is removes while edges are located at the same time.

1.2 Noise Cancellation

Mean noise filter

For this algorithm I recycled code from the previous section (1.1), which resulted in having similar the import statements, main method, and convolving operation. For the mean filter, I initialised a 2D 3x3 kernel as the mask, with all values containing 1/9, as recommended by the relevant COMP422 lecture. I applied the convolve operation on this kernel with the grainy input image and saved the resulting image in a JPG file after converting it to an 8-bit unsigned integer to reduce information loss.

Median noise filter

To implement it this technique, I began by extending the entire image's size by 2 rows and columns (one for each side of the matrix) and assigned zero to these new values. I then created a new image by running through the input image entry by entry, replacing each element with the median of neighbouring elements. I decided on a 3x3 pixel window with which to slide over the entire signal until a new filtered image is outputted. This image is finally converted to a less lossy format and saved in the JPG format within the current directory.

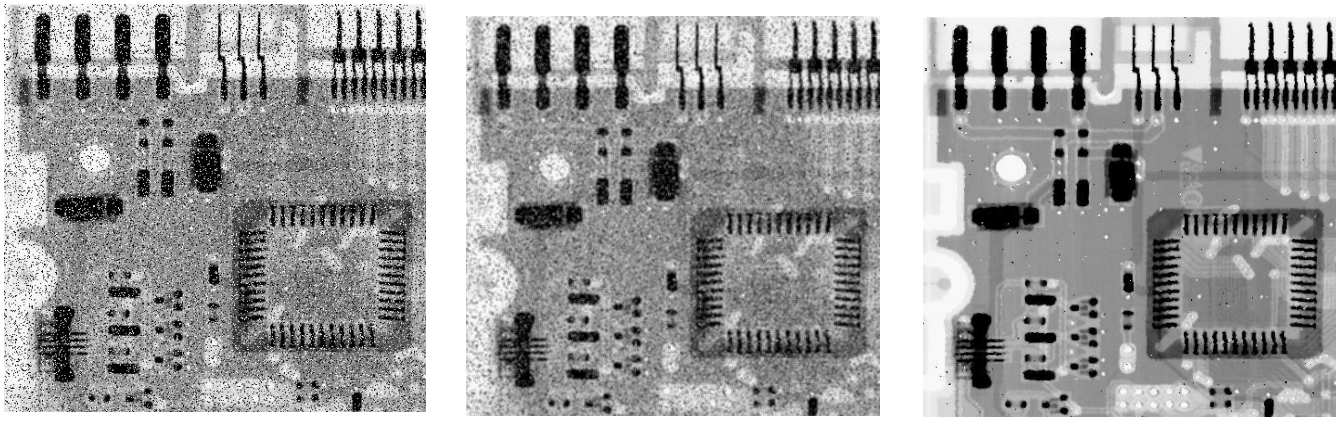


Figure 2. Noisy original image of circuit board (left), and two noise reduction techniques: mean noise filtering (centre) and median noise filtering (right).

Figure 2 illustrates the effects of the filters on the sample image. The effects of the median filter appear to be much smoother than that of the mean filter. This can be explained by the fact of a median being a more robust average than the mean, so a single very unrepresentative pixel in a neighbourhood will not affect the median value significantly. One key difference between these filters is that the median filter can replace the central pixel of the window, resulting in that pixels of contrasting brightness within a neighbourhood being cancelled out. This leaves a less noisy image after filtering compared to the mean filter.

1.3 Image Enhancement

As with the previous two tasks, I recycled the import statements, convolution algorithm, and main method. As explained, the convolution process that I used involves overlaying the mask

on the original image; performing the vector inner product operation, sliding the mask across the image, and finally performing the operation at each pixel location. The summed result of the window replaces the value of the central pixel. I chose to implement a Laplacian filter with a 3x3 kernel with the configuration of [[0, -1, 0], [-1, 5, -1], [0, -1, 0]], which appeared to sharpen the blurry image based on the results (Fig. 6). The Laplacian filter works by compute the second derivatives of an image, measuring the rate at which the first derivatives change.

I chose this configuration because it gave me the best results from all of the tests that I tried – those tests involved changing the central value, looking as online resources, and referring to lecture content. I defined "best" as what looked the least blurry to me when compared sided by side with each other configuration.

Figure 3. Comparison of original blurry image (left) and enhanced image after the application of a Laplacian filter (right).

MINING IMAGE DATA

2.1 Mining Space Images

To implement a method to more easily discern the number of galaxies in the Hubble Space image, I applied a mean filter to smooth the image before applying thresholds of varying values to different maps. I applied a 3x3 kernel and a 5x5 kernels to the image to compare their effectiveness of smoothing, before deciding that that the 3x3 filter was more effective. The former contained values of only $1/9^{th}$, while the latter contained values of only $1/25^{th}$. I found that use of the 3x3 convolutional mask was the most appropriate size for this task because most of the noise was reduced and some of the resolution of the galaxies was preserved compared to the 5x5 mask, as seen in Figure 4.

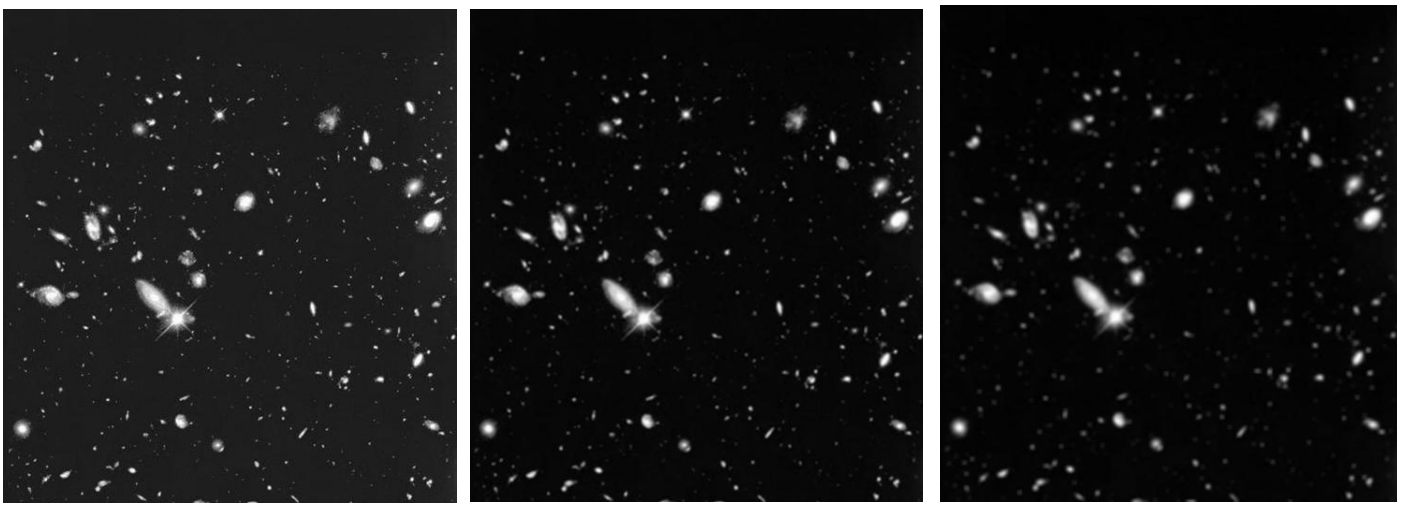


Figure 4. Original image (left), the mean filtered image with a 3x3 kernel (centre) and the mean filtered image with a 5x5 kernel (right). Filtering with the 3x3 convolution mask appeared to be the least blurry of the noise reduced versions.

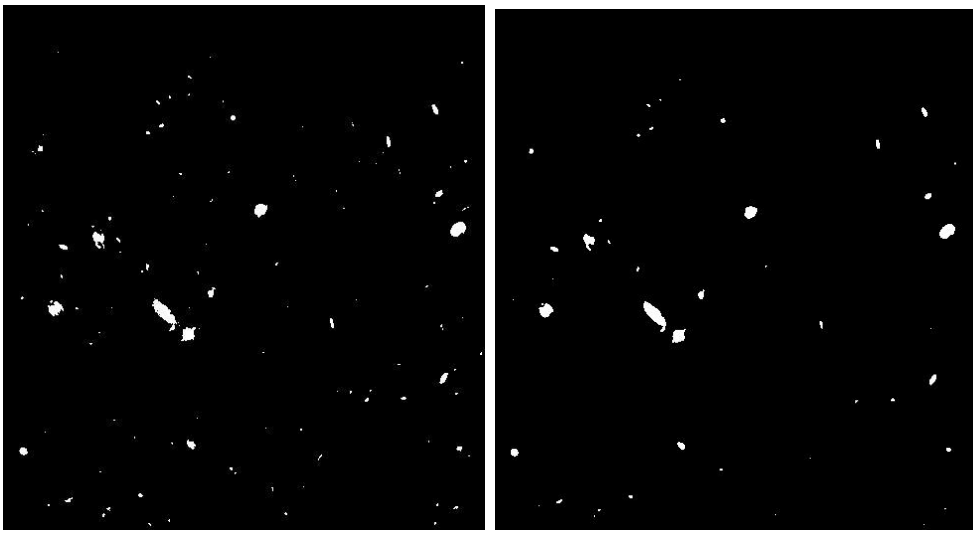


Figure 5. Thresholded images with a value of 200, shown without a mean filter (left) and with a mean filter (right). The filtered image appears to have fewer galaxies.

Thresholding helps to visualise the number of galaxies present above or below a certain brightness. When thresholding is applied without smoothing the image, I noticed the output shows many white dots that would look like galaxies because of the large amount of noise that the input image contained. This is a problem because it makes counting the number of galaxies in the image impossible due to giving false positives and inflating the number of galaxies that are actually present. This is demonstrated in Figure 5. Smoothing helps to reduce the differences in brightness so that light and dark pixels from noisy images becomes the average of the pixels surrounding it.

A threshold in image mining is the maximum brightness of a pixel to class it as white in the outputted black and white binary image. I found that the size of the threshold was directly proportional to the brightness of the galaxy that could be detected – for instance a small threshold like 50 would include galaxies that were big and small, while a larger threshold of

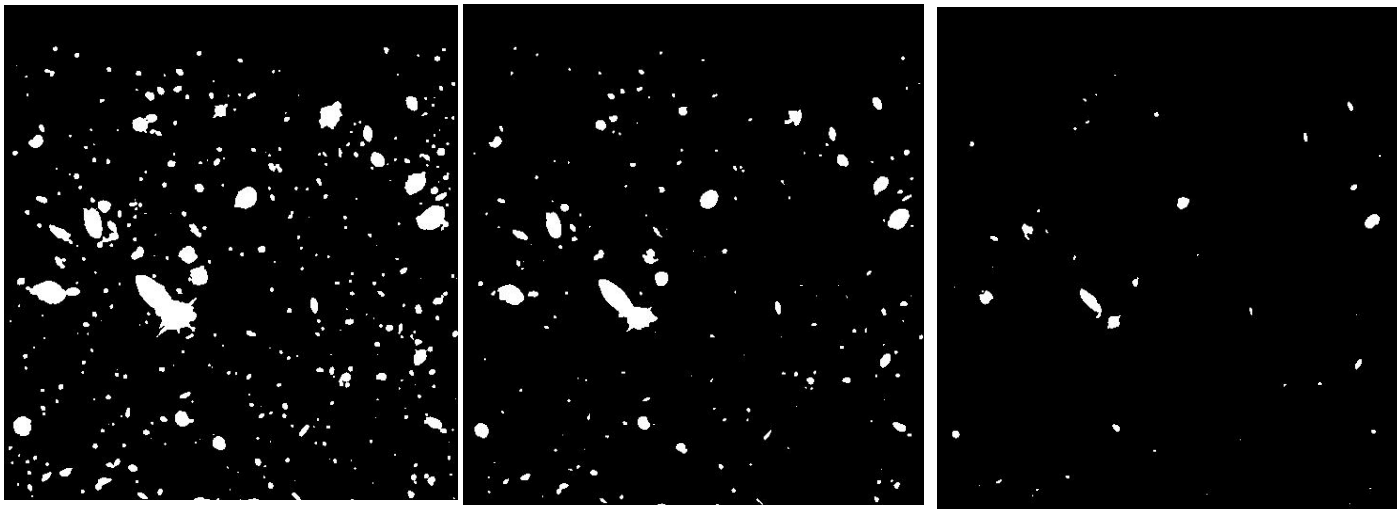


Figure 6. Mapped and filtered original images using a 3x3 convolution mask with a threshold value of 50 (left), 100 (centre), and 200 (right). As the threshold value increased, only brighter galaxies are included in the image, as dimmer ones are filtered out.

200 would only include relatively bright galaxies. These results are visualised in Figure 6 below.

2.2 Face Detection

Feature Extraction

For this task, I decided on ten features to extract from the faces in the given PGM files. These nine features, their extraction method, image section, and matrix size are outlined in the table below. This summarizes the code that is written for this subtask and implementation of these features was inspired by the following article: https://towardsdatascience.com/detecting-face-features-with-python-30385aee4a8e

Facial feature of interest	**Method of extraction**	**Location of extracted feature in image**	**Kernel size**
Pupil colour difference	Absolute value of the difference of matrix minimums	Upper right and left	9x9
Eye white colour difference	Absolute value of the difference of matrix maximums	Upper left and right	9x9
Nose tip brightness	Matrix maximum	Middle	5x5
Nose tip location	Index of matrix maximum mod 5	Middle	9x9
Difference of central information	Absolute value of the difference of matrix means	Lower left and right	9x9
Horizontal and vertical information differences	$\lvert(1/16)\text{sum}(x_{ij}-(1/16))\text{sum}(x'_{ij})\rvert$	Lower left and right	9x9
Diagonal information differences	$\lvert(1/16)\text{sum}(y_{ij}-(1/16))\text{sum}(y'_{ij})\rvert$	Lower left and right	9x9
Other information differences	$\lvert(1/48)\text{sum}(x_{ij}-(1/48))\text{sum}(x'_{ij})\rvert$	Lower left and right	9x9
Class (1 = face, 0 = non-face)			

Creating Tabular Data Sets

I chose to create CSV files from my extracted features in the test and training sets, before converting these to the ARFF format for future use. ARFF is the file format used by Weka data mining software, so that further analysis can take place.

In the file named “train”, 1450 face images and 1450 non face images are converted to 2900 instance vectors, including 9 extracted features and 1 class label. For the class label, a value of 1 signifies that the image has been classed as a face, while a 0 value signifies that the image is not a face.

In the file named “test”, 1451 face images and 1451 non face images are converted to 2902 instance vectors, including 9 extracted features and 1 class label. For the class label, a value of 1 signifies that the image has been classed as a face, while a 0 value signifies that the image is not a face.

Object Classification and Performance Evaluation

I split the testing data and training data 50:50 split because I found that more testing data – that is, a more even split between training data and testing data – gave better accuracy on testing results. I was willing to accept this benefit despite the risk of having a worse model overall as result of having a reduced training data set. Furthermore, a specification of the next section of this report (Section 3.1) was a training/testing split of 50:50, so consistency in this report is preserved by using this 50:50 split.

I used the naive Bayes in WEKA, a machine learning tool created by the University of Canterbury, to evaluate the object classification and performance evaluation on test data and the results are as follows. Number pairs are for training and testing data, respectively.

Attribute	Mean	Standard dev.	Weighted sum	Precision
pupil_dif	17.041 11.2468	27.7719 12.4093	1450 1450	1.877 1.877
eyewhite_dif	20.6491 13.3369	29.7214 14.7488	1450 1450	1.6894 1.6894
upper_dif	1541.7196 1477.4747	2023.1393 1290.8671	1450 1450	8.5198 8.5198
noselip_bright	128.0087 202.6968	67.0354 46.1088	1450 1450	1.0159 1.0159
noselip_loc	1.7697 2.1172	1.6058 0.6998	1450 1450	1 1
bc_dif	29.7952 26.9809	38.1838 23.1606	1450 1450	1.478 1.478
bx_dif	11.8936 10.9389	15.9702 8.9051	1450 1450	0.1772 0.1772
by_dif	10.8113 12.8095	13.7685 10.5644	1450 1450	0.1635 0.1635
bz_dif	28.154 29.5328	37.2841 23.6083	1450 1450	0.1479 0.1479

=== Summary ===

Correctly Classified Instances	78.532 % (2279 instances)
Incorrectly Classified Instances	21.468 % (623 instances)
Kappa statistic	0.5706
Mean absolute error	0.239
Root mean squared error	0.4027
Relative absolute error	47.7922 %
Root relative squared error	80.5393 %
Total Number of Instances	2902

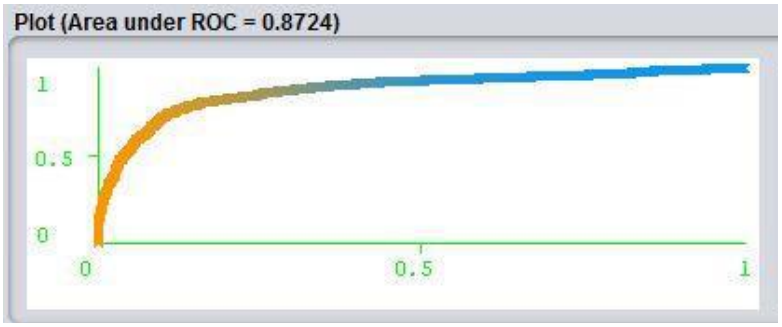
=== Detailed Accuracy By Class ===

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
0 (non-face)	0.700	0.129	0.844	0.700	0.765	0.579	0.872	0.829
1 (face)	0.871	0.300	0.744	0.871	0.802	0.579	0.872	0.883
Weighted average	0.785	0.215	0.794	0.785	0.784	0.579	0.872	0.856

=== Confusion Matrix ===

	Non-face	Face
Non-face	1015	436
Face	187	1264

From the confusion matrix, I calculated the true positive fraction to be 84.4%, from the formula TPF = TP/(TP+FN). Substituted values were 1015/(1015+187).
The overall classification accuracy on training data is 78%.
The true positive fraction (TPF) of my model on the training data and test data are approximately 85% each, and the overall classification accuracies are also close to 80%.
The ROC curve is depicted to the right, and was calculated in WEKA.



According to the information above, the AUC (area under the ROC curve) is 0.872, which means the classifier has a relatively good ability to discriminate faces. This is visualised in Figure 8 where orange represents detection of a face and 0 represents detection of a non-face image.

I ensured that the attributes detected by my classifier were a mixture of domain-specific and general statistical features, so that the classifier was able to generalise better and discern to many types of faces and non-faces.

I would have liked to have more time to use different methods to extract more facial features, for example seeing the effect of smaller pixel box features (e.g. 2x3 windows) and/or the squared difference between the top left and top right quadrants. I did spend some time exploring methods to find more general features, rather than specific features such as the eyes and noses, for example each face's relative symmetry, mean value of the face, and entropy fo the faces. These results ended up being less reliable and unique than the specific facial features that I measured so I ended up excluding these findings from the final report.

DATA MINING USING EXTRACTED FEATURES

3.1 Object Recognition: Classification of Hand-Written Digits

For this program, the main method in this program writes the contents of the extracted feature files to a CSV file. The second method tabulates the input data by opening the

extracted feature files and assigning the contents to a corresponding variable, before appending the data stored in those variables into one long list and returning it.

The tabular data set is created as ARFF format to further steps, and the file is named as ‘numdata’. 649 attributes in 6 feature sets and 1 class label are converted to 2000 instance vectors.

I used WEKA to train a J48 decision tree classifier and test it on the test data, the training data and test data are split from initial data set randomly, where each set contains 1000 instance vectors. Using the confusion matrix in Figure 7, I calculated that the test data’s classification accuracy was 92.4%. Thirty-eight features were found to be in the decision tree, and these were used for the prediction. The constructed decision tree is shown below.

Class	0	1	2	3	4	5	6	7	8	9
0	115	0	0	2	0	1	0	0	0	0
1	0	91	2	1	0	1	0	1	0	0
2	0	2	80	2	0	0	0	2	1	1
3	0	3	1	99	1	8	0	1	0	0
4	0	4	0	2	77	0	0	0	0	0
5	0	0	1	7	0	87	0	0	0	0
6	0	1	0	1	1	0	98	1	0	0
7	0	1	5	4	0	0	0	93	0	0
8	0	0	0	0	0	0	3	0	104	0
9	0	2	0	1	0	0	0	0	1	91

Figure 7. Confusion matrix for the J48 decision tree after being trained for various extracted feature sets on the MNIST data set.

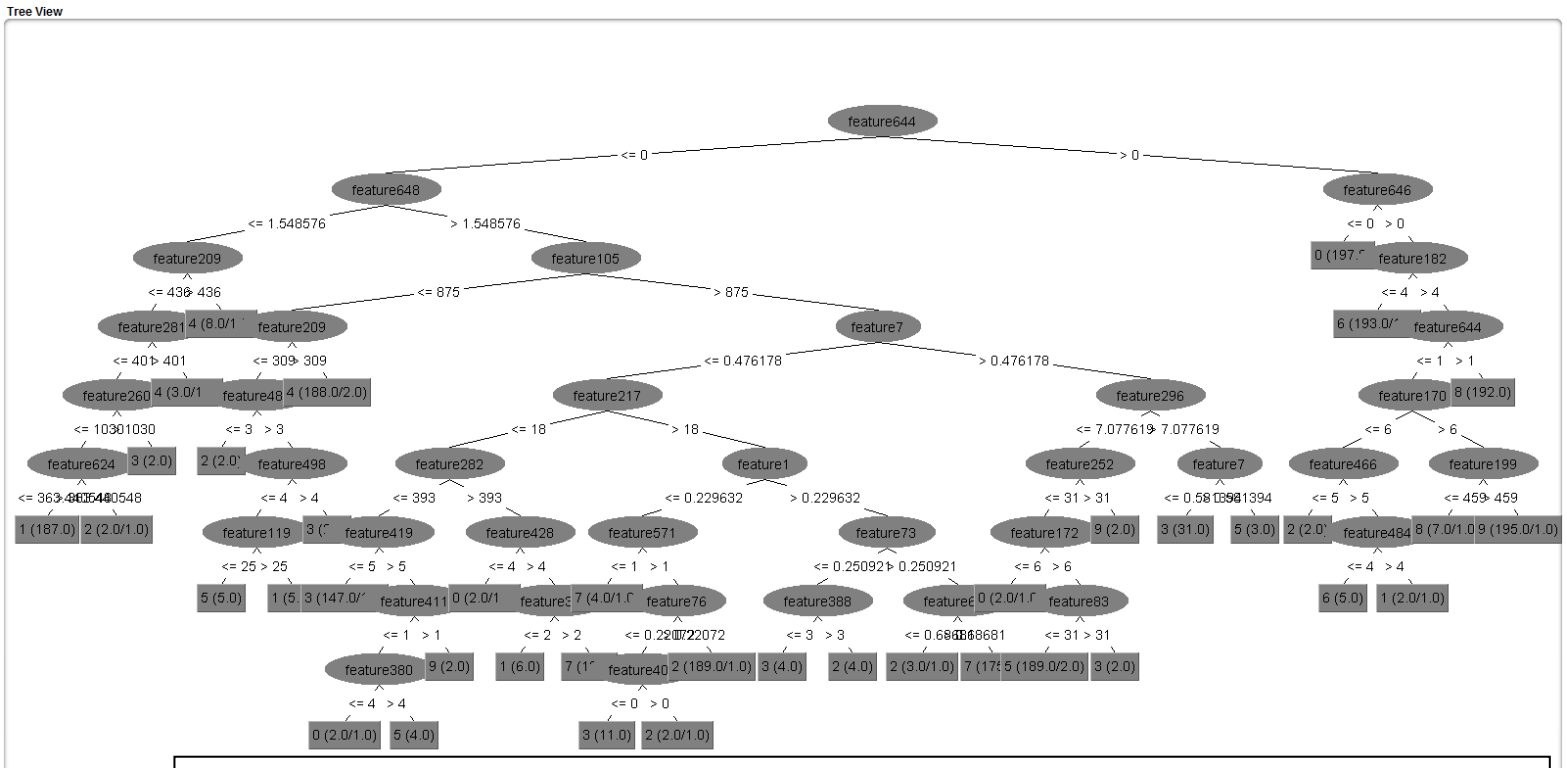


Figure 8. J48 Decision Tree, trained with the six feature extraction files of the MNIST handwritten digit dataset.

Next, another J48 decision tree classifier was trained by using only the 6 morphological features. The training and test data were also split from initial data set randomly, though both data sets were programmed to contain 1000 instance vectors each.

This resulted in the following confusion matrix in Figure 9. More information about this test is included in the Appendix. From the confusion matrix, I calculated the accuracies of each class and visualised them in Figure 8. Appendix C shows the working for the bar graph in Figure 8.

Class	0	1	2	3	4	5	6	7	8	9
0	115	0	3	0	0	0	0	0	0	0
1	0	89	0	1	1	0	0	5	0	0
2	0	1	55	7	4	15	0	4	0	2
3	0	1	11	37	31	25	0	8	0	0
4	0	5	3	4	56	3	0	12	0	0
5	0	1	21	6	5	60	0	2	0	0
6	0	1	0	1	2	0	0	0	0	98
7	0	4	3	3	6	0	0	87	0	0
8	0	0	0	0	0	0	0	0	105	2
9	0	1	0	0	1	1	0	0	0	92

Figure 9. Confusion matrix for the J48 decision tree after being trained on six morphological features on the MNIST data set.

I have visualised the classification accuracy of each handwritten number according to my test data in Figure 10. Five of the handwritten numbers (0, 1, 7, 8 and 9) have an accuracy are greater than 80%, indicating that they are relatively easy to predict according to my model, when compared to the remaining 5 classes (2, 3, 4, 5, 6).

The accuracy of decision tree classifier using all features is 92.4%, which is far above the accuracy of decision tree classifier using only morphological features (69.6%). This result demonstrates that distinguishing hand-written digits only according to morphological features is insufficient to attain the same levels accuracy as using all the features. The accuracy of digit 6 from this classifier is 0, and 96.1% (98 / 102) of its instances are classified as 9. This is likely to be because 6 and 9 re morphologically identical but rotated 180 degrees. From these results, more types of features are required to accurately distinguish hand-written digits from the MNIST dataset.

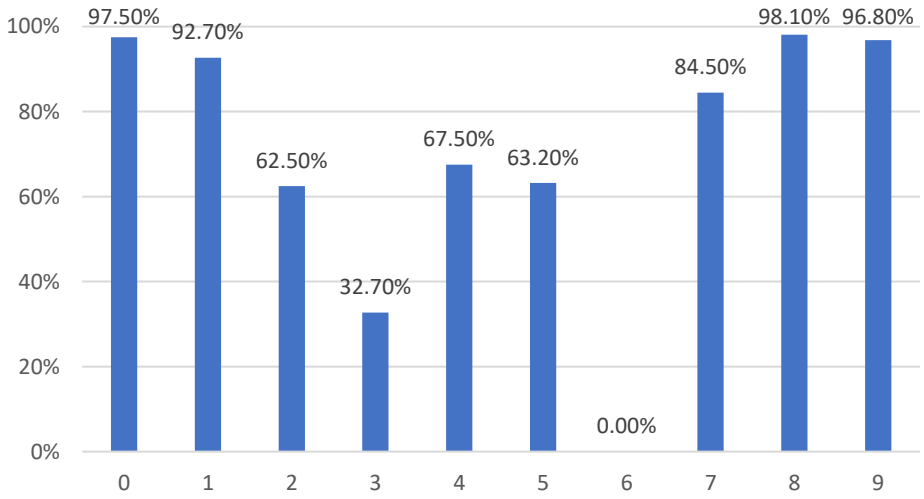


Figure 10. Classification accuracy for each handwritten number in the MNIST Test Data. 1000 instance vectors were used for each training and testing set.

Appendix A

Below is the configuration of the J48 Tree in a breadth-first fashion.

```
feature644 <= 0
| feature648 <= 1.548576
| | feature209 <= 436
| | | feature281 <= 401
| | | feature260 <= 1030
| | | | | feature624 <= 363.440548: 1 (187.0)
| | | | | feature624 > 363.440548: 2 (2.0/1.0)
| | | | feature260 > 1030: 3 (2.0)
| | | feature281 > 401: 4 (3.0/1.0)
| | feature209 > 436: 4 (8.0/1.0)
| feature648 > 1.548576
| | feature105 <= 875
| | | feature209 <= 309
| | | | feature487 <= 3: 2 (2.0)
| | | | feature487 > 3
| | | | | feature498 <= 4
| | | | | | feature119 <= 25: 5 (5.0)
| | | | | | feature119 > 25: 1 (5.0)
| | | | | feature498 > 4: 3 (2.0)
| | | feature209 > 309: 4 (188.0/2.0)
| | feature105 > 875
| | | feature7 <= 0.476178
| | | | feature217 <= 18
| | | | | feature282 <= 393
| | | | | | feature419 <= 5: 3 (147.0/1.0)
| | | | | | feature419 > 5
| | | | | | | feature411 <= 1
| | | | | | | | feature380 <= 4: 0 (2.0/1.0)
| | | | | | | | feature380 > 4: 5 (4.0)
| | | | | | | feature411 > 1: 9 (2.0)
| | | | | feature282 > 393
| | | | | | feature428 <= 4: 0 (2.0/1.0)
| | | | | | feature428 > 4
| | | | | | | feature375 <= 2: 1 (6.0)
| | | | | | | feature375 > 2: 7 (19.0)
| | | | feature217 > 18
| | | | | feature1 <= 0.229632
| | | | | | feature571 <= 1: 7 (4.0/1.0)
| | | | | | feature571 > 1
| | | | | | | feature76 <= 0.22072
```

Continued…

```
| | | | | | | | feature402 <= 0: 3 (11.0)
| | | | | | | | feature402 > 0: 2 (2.0/1.0)
| | | | | | | feature76 > 0.22072: 2 (189.0/1.0)
| | | | | feature1 > 0.229632
| | | | | | feature73 <= 0.250921
| | | | | | | feature388 <= 3: 3 (4.0)
| | | | | | | feature388 > 3: 2 (4.0)
| | | | | | feature73 > 0.250921
| | | | | | | feature626 <= 0.68681: 2 (3.0/1.0)
| | | | | | | feature626 > 0.68681: 7 (175.0)
| | | feature7 > 0.476178
| | | | feature296 <= 7.077619
| | | | | feature252 <= 31
| | | | | | feature172 <= 6: 0 (2.0/1.0)
| | | | | | feature172 > 6
| | | | | | | feature83 <= 31: 5 (189.0/2.0)
| | | | | | | feature83 > 31: 3 (2.0)
| | | | | feature252 > 31: 9 (2.0)
| | | | feature296 > 7.077619
| | | | | feature7 <= 0.581394: 3 (31.0)
| | | | | feature7 > 0.581394: 5 (3.0)
feature644 > 0
| feature646 <= 0: 0 (197.0)
| feature646 > 0
| | feature182 <= 4: 6 (193.0/1.0)
| | feature182 > 4
| | | feature644 <= 1
| | | | feature170 <= 6
| | | | | feature466 <= 5: 2 (2.0)
| | | | | feature466 > 5
| | | | | | feature484 <= 4: 6 (5.0)
| | | | | | feature484 > 4: 1 (2.0/1.0)
| | | | feature170 > 6
| | | | | feature199 <= 459: 8 (7.0/1.0)
| | | | | feature199 > 459: 9 (195.0/1.0)
| | | feature644 > 1: 8 (192.0)
Number of Leaves : 39
Size of the tree : 77
```

APPENDIX B

Training using the six morphological features:

Correctly classified instances	69.6% (696)
Incorrectly classified instances	30.4% (304)
Kappa statistic	0.6626
Mean absolute error	0.02027
Root mean squared error	0.2027
Relative absolute error	36.8514
Root relative squared error	67.4505%
Total number of instances	1000

=== Detailed Accuracy By Class ===

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
0	0.975	0	1	0.975	0.987	0.986	0.987	0.978
1	0.927	0.015	0.864	0.927	0.894	0.884	0.977	0.877
2	0.625	0.045	0.573	0.625	0.598	0.558	0.868	0.487
3	0.327	0.025	0.627	0.327	0.43	0.407	0.816	0.475
4	0.675	0.055	0.528	0.675	0.593	0.556	0.875	0.523
5	0.632	0.049	0.577	0.632	0.603	0.56	0.909	0.515
6	0	0	0	0	0	0	0.929	0.492
7	0.845	0.035	0.737	0.845	0.787	0.763	0.95	0.726
8	0.981	0	1	0.981	0.991	0.99	0.998	0.988
9	0.968	0.113	0.474	0.968	0.637	0.635	0.935	0.465
Average	0.696	0.032	0.649	0.696	0.658	0.64	0.926	0.663