



Cardiff University  
School of Computer Science & Informatics

---

# **Handwriting Recognition Using Image Processing**

*CM3203*

*One Semester Individual Project*

*40 Credits*

*Author*

Kevyne Selmo

*Supervisor*

Dr. Hantao Liu

*Moderator*

Dr. Alia Abdelmoty

# Abstract

The principles of a handwriting recognition system adopt the concept of OCR, but with additional properties to consider, such as the formatting and the style of handwritten characters. The approaches conducted from current systems are based specific ways on how to tackle the problem, which emphasises that there are numerous solutions that could be adopted. The idea of these systems is to resolve the problem of a repetitive task of having to computerise data.

This project aims to build an offline handwritten character recognition system that consisted of multiple, distinct stages: pre-process, segmentation, post-process, feature detection, label training/testing data and classification. These stages cover the details based on the revolving problems of computer vision systems, such as noise reduction techniques and amending fragmented characters, as well as problems specific to the project. Additionally, the system uses three different types of features and classifies them as vectors into a multiclass SVM.

The overall performance of the classification scheme in the system had managed to achieve more than 70% accuracy rate using a combination of the three features implemented into the system. The system is by no means perfect, but the future improvements are discussed in the report, which includes experimenting different features and classifiers.

# **Acknowledgements**

I would like to thank my supervisor, Dr. Hantao Liu, for professionally guiding me throughout the process of this project. It was a pleasure working with someone who had keen interest to the project; the unwavering support contributed into the success of the project.

I would also like to thank my family and friends for providing their support and believing in me.

# Table of Contents

<b>1. Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Approach	1
1.3 Scope	2
1.4 Broad Summary of Outcomes	3
<b>2. Background &amp; Research</b>	<b>4</b>
2.1 Shape Matching and Object Recognition Using Shape Contexts	4
2.2 User Acceptance of Handwritten Recognition Accuracy	4
2.3 A full English sentence database for off-line handwriting recognition	5
2.4 Recognition of Handwritten Numerals with Multiple Feature and Multistage Classifier	6
2.5 Handwritten English Character Recognition Using Neural Networks	7
2.6 Support Vector Machine (SVM) for English Handwritten Character Recognition	8
<b>3. Design</b>	<b>10</b>
3.1 Specifications	10
3.1.1 Pre-process the images	10
3.1.2 Locate the regions in the images	10
3.1.3 Extract features to uniquely identify each character	10
3.1.4 Label the sample data	11
3.1.5 Classify the extracted features	11
3.2 Flow Chart	12
3.3 UML	14
3.4 Pseudocode	15
3.4.1 Pre-process	15
3.4.2 Segmentation	15
3.4.3 Post-process	15
3.4.4 Feature Detection	15
3.4.5 Label Training Data	15
3.4.6 Classification Scheme	16
<b>4. Implementation</b>	<b>17</b>
4.1 Pre-process	17
4.1.1 Problem – Noise	18
4.1.2 Problem – Threshold	18
4.2 Segmentation	20
4.3 Post-process	20
4.3.1 Problem – Fragmented Characters	21
4.4 Feature Detection	22
4.5 Classification	23
4.6 Labelling Sample Data	24
4.6.1 Problem – Label Training Data	24
<b>5. Unit Testing</b>	<b>25</b>
5.1 Pre-process	25

5.1.1 Filtering Noise	25
5.2 Segmentation	29
5.3 Post-process	30
5.4 Feature Detection	32
5.4.1 Density	33
5.4.2 Centroid	34
5.4.3 Orientation	35
5.5 Labelling Training Data	36
5.6 Classification	37
<b>6. Evaluation &amp; Results</b>	<b>38</b>
<b>7. Future Improvements</b>	<b>44</b>
7.1 Larger sample size	44
7.2 Consecutive image processing for sample data	44
7.3 Adaptive thresholding	44
7.4 Algorithm for joining fragmented characters	45
7.5 Features: new features and performance	45
7.6 Classification scheme	46
7.7 Store the classification scheme	46
7.8 User Interface	47
<b>8. Conclusion</b>	<b>48</b>
<b>9. Reflection</b>	<b>51</b>
<b>10. Appendix A</b>	<b>53</b>
<b>11. Appendix B</b>	<b>54</b>
11.1 Density Vector	54
11.2 Average Distance Vector	54
11.3 Orientation Vector	54
<b>12. Appendix C</b>	<b>55</b>
12.1 000.png	55
12.2 001.png	56
12.3 002.png	57
12.4 003.png	58
12.5 004.png	59
12.6 005.png	60
12.7 006.png	61
<b>13. References</b>	<b>62</b>

# Table of Figures

Figure 1: Form filled in as my sample data.	6
Figure 2: Example binary representation of characters.	8
Figure 3: Shows a distinguishable property of the character.	8
Figure 4: Example representation of that the grid should look when applied to a character.	11
Figure 5: Flow chart of the system.	13
Figure 6: UML diagram representation of the system.	14
Figure 7: Median filter implemented after image had been converted as grayscale.	17
Figure 8: Algorithm that inverts the colour of the pixels.	18
Figure 9: Histogram of 002.png (See Appendix C).	19
Figure 10: Closer look into the gray pixels.	19
Figure 11: Using connectivity algorithm and regionprops to obtain information about objects in the image.	20
Figure 12: Modifying the unwanted regions saved by regionprops.	20
Figure 13: Example of fragmented handwritten character.	21
Figure 14: Algorithm which dilates the image and the result from dilating the character 'I'.	21
Figure 15: Before (with dilate)	22
Figure 16: After (with dilate and algorithm).	22
Figure 17: The character 'A' divided into cells to represent a grid.	22
Figure 18: Neuburger's multi-class SVM.	23
Figure 19: Displaying the classified image.	23
Figure 20: Display the accuracy against the test data.	24
Figure 21: Snippet of the sample data.	24
Figure 22: Code snippet of the algorithm to label the sample data.	24
Figure 23: Image before filtering.	27
Figure 24: Image after filtering.	25
Figure 25: Image with median filter.	28
Figure 26: Image with bwmorph (majority).	26
Figure 27: Image before border cleaning.	29
Figure 28: Image after border cleaning.	27
Figure 29: Example of character after applying 238/256 threshold.	28
Figure 30: Example of character after applying 245/256 threshold.	28
Figure 31: Image with 238/256 threshold.	28
Figure 32: Example of character after all noise reduction techniques.	29
Figure 33: Accumulate the number of characters in one sample set and displaying the results.	29
Figure 34: Displaying the detected regions.	30
Figure 35: Example of a fragmented character, width wise.	32
Figure 36: Example of a fragmented character fixed.	30
Figure 37: Before and after fixing the regions.	31
Figure 38: Example of fragmented character, height wise.	33
Figure 39: Example of fragmented character fixed.	31
Figure 40: Before and after fixing the regions.	31
Figure 41: Example of how algorithm traverses from cell to cell.	32
Figure 42: Visual example of the algorithm applying a grid into the character.	33

Figure 43: Original image.	35
Figure 44: Visual grid with labelled cells.	33
Figure 45: Displays the information of cell number 1 in terms of density.	34
Figure 46: Displays the information of cell number 1 in terms of centroid.	34
Figure 47: Cell number 1 of the character in Figure 44.	35
Figure 48: Edited sample data.	38
Figure 49: Results displaying as expected.	36
Figure 50: Edited sample data.	38
Figure 51: Results from algorithm.	36
Figure 52: Sorted results.	37
Figure 53: Results after classifying the image, similar to the training data.	37
Figure 54: Classifying image 001.png against 002.png.	37
Figure 55: Original image.	40
Figure 56: Image after 238/256 threshold.	40
Figure 57: Image after 150/256 threshold.	38
Figure 58: Before post-processing stage.	41
Figure 59: After post-processing stage.	39
Figure 60: Without additional features.	42
Figure 61: With additional features.	40
Figure 62: Performance of classification scheme.	40
Figure 63: Snippet from the sample data.	41
Figure 64: Test results with accumulating training samples.	42

# 1. Introduction

Handwriting recognition systems had been developed in the fields of computer vision for many years, where these systems had dissimilar approach to one another <sup>[1]</sup> <sup>[2]</sup>. In other words, there are a multitude of approaches that could be used for this problem. These approaches include trying to recognise handwritten words instead of characters, whereby the process to do this task differs completely when trying to recognise characters individual.

The basis of this project is to recognise individual characters instead of recognising handwritten words <sup>[3]</sup> <sup>[4]</sup> as a whole. The project extrapolates on the challenges faced when developing such a system. In particular, the project's main focus is not entirely on the classification of the system, but more about the process of getting to the classification stage.

The majority of the current articles cover mainly the performance of their system, but this project will cover the importance of gathering and preparation of data because I believe that the performance of the overall system would be affected when using the appropriate procedures.

In this thesis, I will be covering my approach of building an offline handwritten recognition application. To be more specific, the recognition system is capable of identifying handwritten characters from an input image by using training data, where the majority of the handwritten characters are correctly classified with high accuracy.

## 1.1 Motivation

From the audience perspective, the application is ideal for automation in terms of being able to recognise user's handwritten input. For example, the application can be used to recognise user's input, such as a form filled out in block capital letters. This would save time instead of manually typing out the user's input. The motivation of the program is to tackle the problem of having to type what had already been written into the computer. The premises of the project lie on the area of OCR (Optical Character Recognition) <sup>[5]</sup>.

## 1.2 Approach

In order to develop my program, I had used MATLAB (Matrix Laboratory) <sup>[6]</sup> to implement my application. The development process of my program is split into multiple, distinct stages. The first stage of my program is the *image acquisition/pre-processing* stage. This stage deals with vital components and procedures that need to be done before attempting to recognise the characters because the procedures removes noise that may distort the results, and it also prepares the image for the processes to come.

The next stage of my program is *segmentation*. The idea of this stage is to locate each individual character that is present on the image, as well as gaining information about the characters, such as the area and centroid. The information about the characters are



essential as they help in many ways such as filtering remnants of noise by using their area or using the centroid to calculate a feature.

Once the characters are located, the program goes through the next stage which is the *post-processing* stage. This stage deals with fragmented characters, where if a character is fragmented, then it will amend the information obtained from the previous stage, which would result in only having one location for the character instead of multiple locations, as well as a new centroid and area. This is done so that the fragmented character is processed as one instead of individual fragments.

Upon completing the stages mentioned above, the image is ready to extract features of the characters detected. The feature detection stage generates a feature vector for all of the characters that are present in the image, which will be used to classify the characters.

However, before classifying the characters, there is a stage called *label training data* stage, which is used to label the training or test data. This is done through a simple algorithm, which is further explained in the implementation section of this thesis. The idea of the algorithm is to label the feature vectors because the classifier needs to know which character of the alphabet is the vectors are supposed to represent.

The classification scheme is the next and final stage of the program, which tries to identify the characters of the user's image input or test data based on the training data. This is done by using a modified SVM (Support Vector Machine) [7] that allows multiple classes classification by creating models which are based on a one against all relations. Since the feature vector can consist of  $n$  dimensions, SVM is suitable for this case as it is capable of supporting multiple dimensions.

### 1.3 Scope

The nature of my thesis may differentiate depending on the perspective of the approach. As part of the scope of my project, I had applied a constraint where I had limited the recognition capabilities to only being able to recognise non-cursive handwritten characters. If the system was meant to recognise cursive characters, then the requirements of the system will differ and will have a completely different approach, whereby it would become meaningless trying to recognise cursive characters, due to the fact that it would be more difficult to segment cursive characters, so the ideal approach is not to do this and instead try to recognise the entire word [4].

On the contrary, I was also restricted with my training and testing sets, where I was unable to find the suitable sample data for the type of approach I had conducted for the project. The initial plan included finding a database containing variations of the English characters/letters, but there was a lack of resources, freely available that were suitable for my approach.

I decided to create and gather my own sample data. The problem with this approach is the time to gather the data. I had to consider the time constraint, which only allowed me to gather a small sample data. The report will include further information about my approach on gathering the training and testing sets.

In addition, my approach had only been applied and tested against the block capital of the English alphabet due to the time constraints of developing the program, but this does not mean that the approach in this thesis will not work for other languages or even digits (0-9). As a speculation, provided that the same approach is taken and minor changes to some of the parameters of the system, such as altering the `labelTrainingData` function, then the program is robust enough to be able to be used for recognising other handwritten characters.

### *1.4 Broad Summary of Outcomes*

Throughout the development of the system, rigorous tests were conducted continuously to ensure that the system provides robustness and accuracy. The end system included being capable of:

- removing/ignoring noise,
- segmenting individual characters,
- dealing with fragmented characters,
- obtaining features from the characters,
- labelling the feature vectors, and
- classifying the characters from the input image.

The program had achieved over 70% accuracy when classifying the test set. However, the program is not able to recognise some invariants; such as if a character is rotated at an angle, then the program would most likely be unable to recognise the character. Some of the invariant cases are extreme, where they are unlikely to happen, so the program has a direct approach to realistic probabilities of human input. However, ignoring the extreme cases, the system is capable of achieving what is classified as a 'good recognition system'.

## 2. Background & Research

### *2.1 Shape Matching and Object Recognition Using Shape Contexts*

Upon conducting my initial research, I came across a paper <sup>[8]</sup>, which measures the similarity against shapes, such as handwritten digits, trademarks, etc. and uses the measurement for object recognition. This is achieved through a set of stages, which involves “solving for correspondences between points on the two shapes” then “using the correspondences to estimate an aligning transform, whilst computing the distance between the two shapes”, which creates a sum of matching errors of the corresponding points to obtain a magnitude of the aligning transformation. They solved this correspondences problem by using a descriptor to each point, and using it to capture the distribution of the neighbouring points, which gives a “globally discriminative characterisation”. Feature points or descriptors are information about the object, such as edges.

In context to my project, I believe that this method is applicable for my project in terms of recognising handwritten characters using their method as a means of collecting feature points. According to their results when experimenting with handwritten digits, they had managed to achieve less than 0.1% error rate of the test set, which consisted of 60,000 training set and 10,000 test set. This is a strong suggestion that their method has a solid foundation that could be used in the development of my handwriting recognition system.

On the contrary, this method requires a generous amount of training and test data, which I may not be able to obtain with the time constraint of the project. This means that this method is difficult to achieve due to the sheer size of sample data that they have, which I do not possess and difficult to find. If I was to use only the data that I’ve gathered myself, then the system may have a high error rate because of the lack of training data.

However, I may be able to incorporate some of the ideas based on this article. For example, the idea of differentiating between shapes using a sum of matching errors could be adopted onto my project and I would instead apply my own method of achieving the sum. This would provide discrimination between the handwritten characters as they all have some unique features, but also similar characteristics.

### *2.2 User Acceptance of Handwritten Recognition Accuracy*

In this article <sup>[9]</sup>, the author, Mary, tries to figure out how accurate does the recognition system has to be considered as useful. The article covers an evaluation of user acceptance of different handwriting recognition rates, which is considered as one of the paramount aspect of the program. The result from the article shows that the participants had considered a recognition rate of 97-99% as their acceptable boundary.

Speaking in general terms, to attain accuracy level that is close to 100% is not a simply task and nearly impossible, but building a recognition system that is capable of

achieving an accuracy that is high enough that is considered as acceptable is the main goal.

Although this article does not necessarily cover the steps or procedures on how to recognise handwritten characters, it does provide an insight to what the scope of the project has to be and provides a realistic point of view towards the development of the program.

To clarify, achieving 100% accuracy of a handwritten recognition system is highly unlikely, if not impossible. The system could only cover a certain amount of invariants to a point where new errors will start to appear that are not caused by the system, but by the human instead. A simple example would be an infant that had only begun learning how to write. Now if the system tries to recognise their handwriting then the likelihood of getting 100% recognition accuracy is low, unless the training data consist of infant handwriting.

In addition, even if you have large amount of training data, there are too many variations to each individual's handwriting. To a certain extent, there is always going to be a percentage of error rates, no matter the size of your training data due to the vast individualism of handwritten characters.

### *2.3 A full English sentence database for off-line handwriting recognition*

This next article <sup>[10]</sup> may not be completely relevant to the functionality of the system, but had given the idea of constructing the database containing a multitude of characters that are structured in order to programmatically label them as the training and test set. The concept in the article to obtain the data based on the forms filled out, where the authors were able to extract the handwritten texts due to the structured nature of the form, i.e. they knew where the handwritten text was and other irrelevant information.

When conducting my research, it was difficult to find the database that I wanted, so as a result, I had decided to make my own. For my project, I will use a structured form (see Appendix A) that will be filled out by numerous anonymous people who'll write the alphabets structurally, as seen in Figure 1. Once the form had been filled in, the document is scanned, and will serve as part of my sample data.

Notice that there is a grid in Figure 1, which contains all of the alphabets, 7 times. I had made the outline of the grid as transparent as possible, so that there won't be too much difficulty when trying to process the image in terms of amplifying the outline as noise. Additionally, by using the grid, it would guide the user to write the characters in the specific position, which would give an easier task when labelling the characters. The structured grid allows me to know where each character is placed and which character it is.

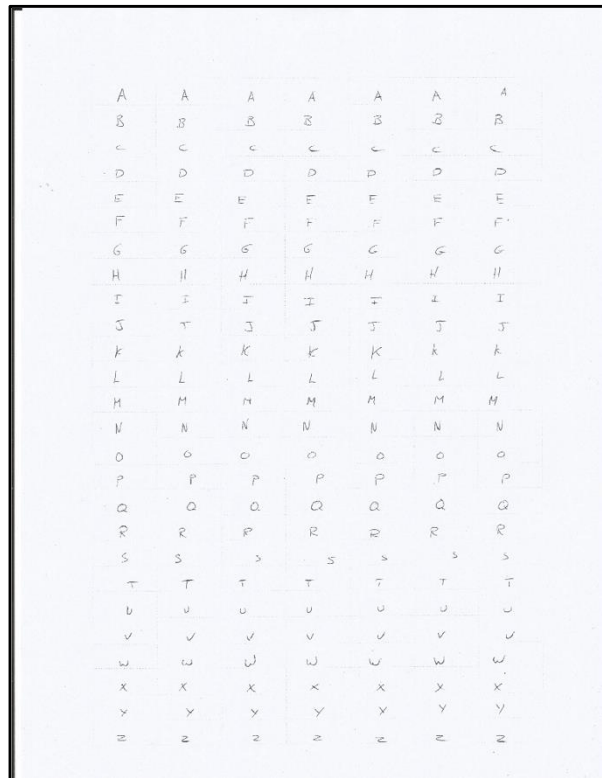


Figure 1: Form filled in as my sample data.

## 2.4 Recognition of Handwritten Numerals with Multiple Feature and Multistage Classifier

In this paper <sup>[11]</sup>, a multiple experts' system that uses neural network is involved in recognising handwritten digits, which includes: "an incremental clustering neural network algorithm" that has merging and cancellation process, feature extraction method using a modified directional histogram, and an algorithm that does "learning rejection neuron strategy". They based their work on "Handwritten numerical recognition based on multiple algorithms" by Kimura and Shridhar <sup>[12]</sup>.

One of the current problems with recent recognition systems is that most of them only use one feature method and one classification scheme. The idea proposed by this paper is that they had used multiple features and classifiers, as well as utilising a divide and conquer subgroup strategy in order to achieve superior results compared to the current systems.

Ideally, to achieve a low error rate, the program should have at least 2 or more features methods. The combination between the features methods would result in a more reliable and accurate classification. This is to be expected compared to a single feature method due to the fact that the other features would enforce the correctness or wrongness of the results given by the classifier. In other words, multiple features and classifiers are used to verify the results, but at the cost of increasing the time and space complexity of their system.

The scope of this project is limited, which means that adopting this recognition method may be difficult to achieve. However, the concept of their idea may be considered during the final weeks of the development stage, which depends on the speed of development of the program as well as the time constraint.

I would like to put emphasis on this section of my project because depending on which feature method is used would depend on how good the results will be. Before implementing additional feature methods, I will comply with just one and thoroughly implement the method to achieve the optimal results from the features gathered. If given more time, then I would attempt to implement different feature methods to further enhance the results of the classifier.

As for multiple classifiers, I would stick to just a single classifier. In this article, they had used two different types of neural networks. From my perspective, their classification scheme is complex and time consuming, where implementing the algorithm for it would take too much time, especially since there are two of them to implement. For this project, I would adopt a simple classifier that would take a small amount of time to implement and prioritise other implementation. This would allow more time to be allocated in creating new feature methods which are likely to improve the results in terms of accuracy because I believe that choosing a classifier will depend on what features are being used.

## *2.5 Handwritten English Character Recognition Using Neural Networks*

In this article, one of the main points is using neural networks that had been trained using an algorithm, “error back propagation” <sup>[13]</sup>, to obtain accurate recognition whilst keeping a low memory usage, where their results show an accuracy of more than 70% just from the back propagation network. Due to the nature of neural networks, multilayer perceptron networks are capable of achieving superior accuracy when it comes to recognition. This is because of the network is constantly changing, where every change would make the network learn or adapt to suffice to new data.

For my project, I may want to adopt the idea of back propagation, but unlike how they have used it in their method. The idea of back propagation for my project is used a verification algorithm that would be used to help the classification scheme to improve its accuracy. However, this is not necessarily a ‘must have’ feature implemented to the program, so if there is time for further implementation then this feature would earn its spotlight.

On a lighter note, the article covers the basic steps that talks about how they had used matrices to represent the characters in their skeletal, binary form, as seen in Figure 2. This idea is useful for my project before extracting the features, since this provides a simple representation of the characters.

First Character 'A'	Second Character 'B'	Third Character 'C'
0 0 1 0 0	1 1 1 1 0	0 1 1 1 1
0 1 0 1 0	1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 1 1 1 0	1 0 0 0 0
1 1 1 1 1	1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 1 1 1 0	0 1 1 1 1

Figure 2: Example binary representation of characters.

Idealistic wise, this representation of binary alphabets is basic, but has the potential to provide good features. As you can see from Figure 3, the characters are distinct, for example, the character 'C' has a unique feature of zeros that are uniformly placed compared to the 'A' and 'B' matrices.

Third Character 'C'
0 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
0 1 1 1 1

Figure 3: Shows a distinguishable property of the character.

However, by converting the image into binary, the image loses information depending on the set threshold. For my program, when processing the image (converting the image into binary), there has to be a fine-balance of this threshold, where having a threshold that is too high would result in losing information about the characters present in the image, but would eliminate most of the unwanted noise, and vice versa.

## 2.6 Support Vector Machine (SVM) for English Handwritten Character Recognition

For this paper, they had proposed handwritten character recognition system with distinguishable stages <sup>[14]</sup>. These stages are: pre-processing, feature extraction, and classification. The first stage is to 'clean' the characters by producing a skeletal representation of the character. The next stage uses FCC, 'Freeman chain code', which provides the boundary of the characters that is used to build a feature vector, totalling up to 64 features. Finally, the last stage involves a SVM to build a classification scheme, where their results had provided a relatively high accuracy of more than 70%.

The concepts and ideas in this paper are applicable to my project. For example, the idea of 'cleaning' the characters before extracting features is a basic step in any recognition system, but also an essential step. My project will need to adopt an algorithm that would process the image, ready for feature extraction. The algorithm will consist of basic

computer vision techniques such as converting the image into binary and/or using a filter for the purposes of noise removal.

Moreover, I believe that their method of extracting features is unclear and rather complicated due to modification of adding a heuristic, which may degrade in terms of their system's performance. For my project, I would like to adopt basic feature extraction methods that are relatively quick to implement, but could be valued as good features. This way, I would be able to adapt the program, so that I will be able to experiment with different types of features and perhaps even a combination of them.

As for their classifier, they had chosen SVM, which I was planning on using initially. Since SVM are constraint to only having a binary choice, e.g. A could either be of type class A or B, and then I would need to modify the classifier into allowing multi-class SVM. One of the main features of SVM is that it can have  $n$  dimensions. This will prove to be useful considering that I would most likely produce a vector consisting of a multitude of values.

When comparing the systems mentioned in the previous articles, they seem to have a contrast approach. For example, Kimura and Shridhar approach was based on using multiple features and classifiers in order to achieve high accuracy rate, whereas the other system focuses more on a single classifier. The results show that having multiple classifiers provide a lower error rate substantially. This emphasises that this type of approach makes a better system, but at a price in terms of the cost of performance in the form of time and space complexity. From this, I had become aware of this trade-off that I will have to consider when building my system.

In summary, upon conducting my research and doing some background readings, I had gained an insight to what are the basic requirements of a recognition system as well as going into the specifics of handwritten recognition systems. The research had provided an outlook to what types of algorithms the system should have, and how to go about solving the proposed problem, such as breaking up the problem into stages that makes the individual parts of the system.



## 3. Design

### 3.1 Specifications

In this section, I will be discussing what the capabilities of the end-system should have, which includes a discussion of the system's functionality. I had included an acceptance criterion for each specification which is the condition(s) that the end-system must satisfy. This will be tested after the implementation stage, to find out if the conditions had been met.

#### 3.1.1 Pre-process the images

In this specification, I will be implementing a function which will:

- Remove noise by using multitude of noise reduction techniques such as applying a median filter <sup>[15]</sup> and bwmorph <sup>[16]</sup>.
- Convert the images from RGB to grayscale <sup>[17]</sup> then binary <sup>[18]</sup>.
- Revert the binary colour, e.g. white to black, vice versa.
- Connect regions which are separated by one pixel, e.g. bridge <sup>[16]</sup>.

*Acceptance Criteria:* The function should be able to remove as much noise as possible whilst being able to keep the important details of the character, where the characters are still be distinguishable after applying all of the image processing techniques. By distinguishable, this means that the handwritten characters can be identified and interpreted correctly by an individual, visually.

#### 3.1.2 Locate the regions in the images

I will conduct research that consists of looking into some MATLAB's built-in functions that detects the position of the characters. A function will be implemented to which will do the following:

- Finds the regions using 8-way connectivity <sup>[19]</sup>.
- Extract information about the regions. Information such as the area and centroid of each region, and smallest possible bounding box that contains each region <sup>[20]</sup>.
- Remove regions that have an area which is not within a specified range.
- For visual purpose, draw the bounding box for each region <sup>[21]</sup>.

*Acceptance Criteria:* The function should be able to locate valid regions in the images and create a struct variable that contains the area, centroid and position of the regions. The struct should not contain regions which are too small or large, since small regions are likely to be remnants of noise and large regions are not considered as characters. Additionally, the valid regions have a bounding box containing them when the image is displayed.

#### 3.1.3 Extract features to uniquely identify each character

For this, I will Implement a function which:

- Extracts features from the regions and saves them into a vector.
- Draw the grid for visual purposes.

These features should be obtained by dividing the regions into an  $N \times M$  grid, where each cell in the grid will provide 3 features which are:

- Density of white pixel,
- Average distance of white pixel from the centroid,
- Orientation of the cell.

Using a combination of features, evaluate them with the classifier to see what level of accuracy the features are able to achieve.

*Acceptance Criteria:* The function should be able to give a vector which will contain the features that had been extracted from all of the characters. Furthermore, grids are draw for each of the detected characters in the image for visualisation purposes.

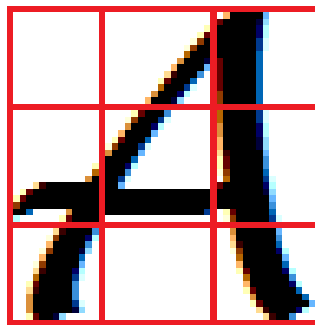


Figure 4: Example representation of that the grid should look when applied to a character.

### 3.1.4 Label the sample data

In this, I will build an algorithm that will:

- identify all of the regions based on their position in the image, and
- determine the correct label, i.e. give the region a corresponding character.

*Acceptance Criteria:* The function should be able to 100% accurately label the regions in the training and the test set with the correct associated character. If 100% accuracy is not achieved, then when it comes to testing, I will not be able to evaluate the effectiveness of the recognition system against the test set, i.e. the outcome is incorrect.

### 3.1.5 Classify the extracted features

I will implement a function that will:

- Use Support Vector Machine [7], in order to classify the extracted feature vectors.
- Display the percentage of correctness when using the training set against the test set.
- Output the classified characters based on the test set or input image.

*Acceptance Criteria:* The function should be able to use a modified SVM which will try to classify  $n$  dimensions of features that are based on 26 classes. When using the test set, the function should provide a percentage of accuracy that will be used for evaluation purposes. The function should provide a text display of the classified characters based on either the test set or the input image.

### *3.2 Flow Chart*

The following diagram is an overview of how the functionality of the system works on a high level representative [22]. The diagram tells how the data (training set, test set, and/or input image) goes through multiple procedures before the system displays the classified data.

Notice in my flow chart design, I had indicated that the training and test sets as databases, but in my approach I had only used several images as my training and test sets. However, the ideal approach is to use a large database of sample data. Due to the time constraint, and other priorities concerning development, I was only able to use a minimalistic, but sufficient number of sample data.

On the contrary, the flow chart also shows that in order to classify the test set or input image, a classification scheme must be built to do the classification. Naturally this is a given, but the visual representation provides a clear and sound understanding of the associations between the test set and the input image against the classification scheme, which is useful when developing the system, as well as the process of creating the classification scheme.

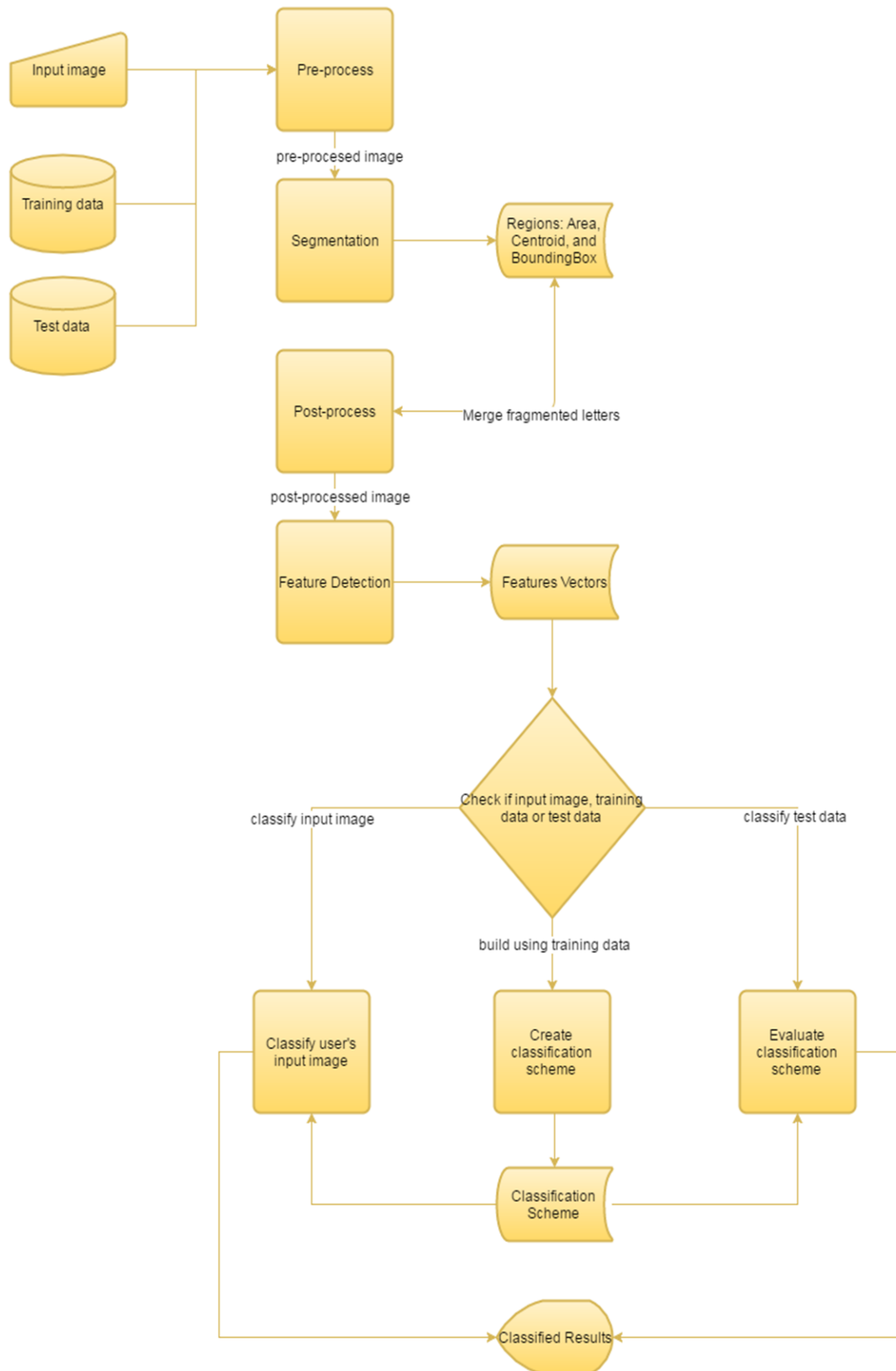


Figure 5: Flow chart of the system.

### 3.3 UML

The following UML class diagram <sup>[23]</sup> provides a full detailed architecture design of my recognition system. The class 'Recognise' is basically the main function which calls the functions from the 'CharacterRecognition' class. In addition, the class maintains most of the important global variables used throughout the process of recognition, such as the struct containing the information of each character in the image.

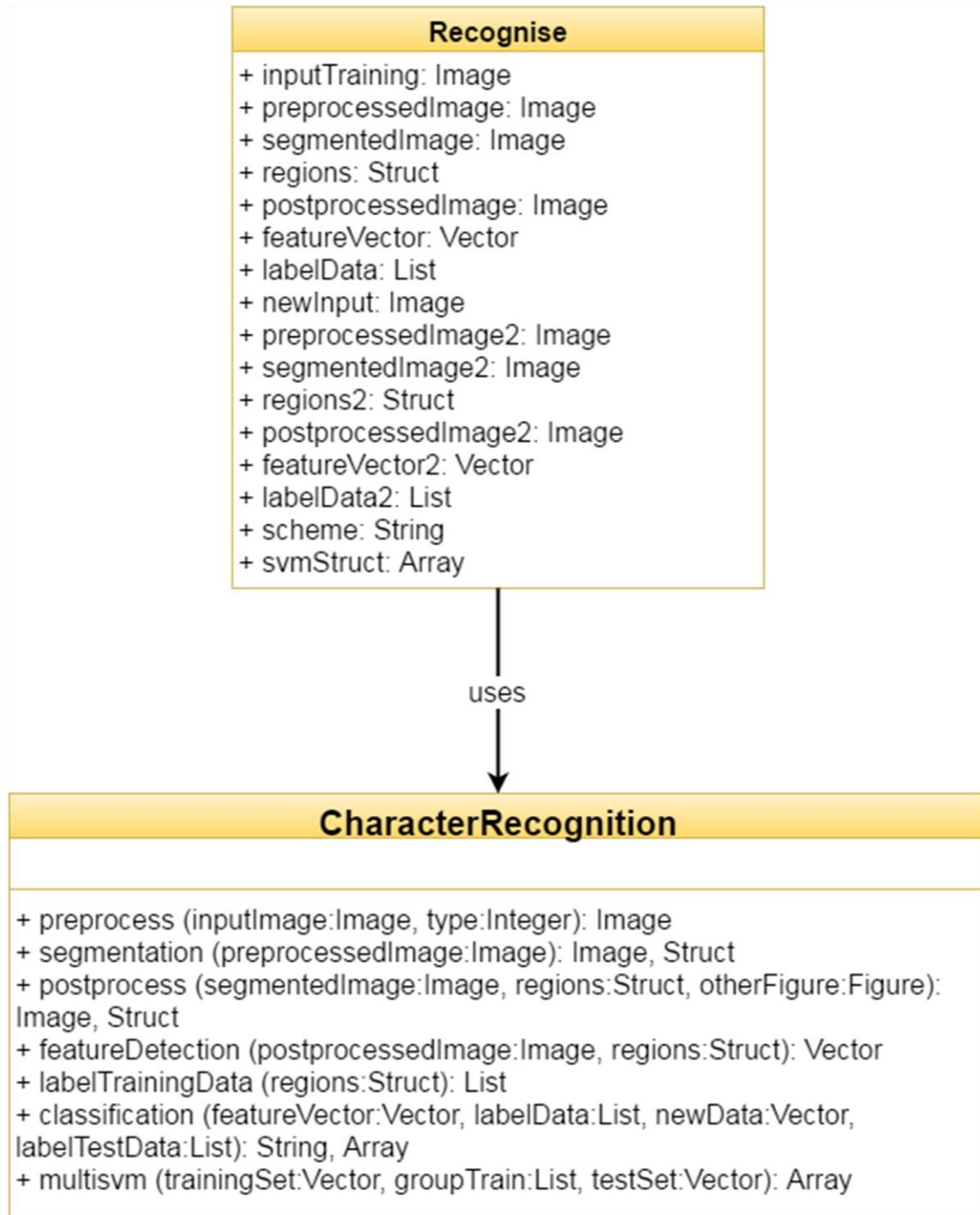


Figure 6: UML diagram representation of the system.

### 3.4 Pseudocode

Below is my pseudocode when developing my recognition system. Before the actual implementation, I had decided to design these following pseudocodes in order to get a basic idea of what I expect the function should do. Additionally, for future developers, the pseudocode will provide a different perspective when trying to understand how the system works, if the comments in the code failed to do so.

*Note: these algorithm segments might not fully represent the end system and should be taken as designs.*

#### 3.4.1 Pre-process

```
1  function preprocess (initialImage)
2      convert image colour to grayscale
3      apply noise filtering algorithm
4      convert image colour to binary
5      dilate image
6  return processedImage
```

#### 3.4.2 Segmentation

```
1  function segmentation (processedImage)
2      use 8-way connectivity to find regions
3      store their coordinates and area
4      place bounding box to regions that have specified area
5  return segmentedImage
```

#### 3.4.3 Post-process

```
1  function postprocess (segmentedImage)
2      find regions that are close to each other
3      merge these regions
4      apply algorithm to produce an outline of each letter
5  return postprocessedImage
```

#### 3.4.4 Feature Detection

```
1  function featureDetection (postprocessedImage)
2      loop through each letter
3      for each letter create an  $N \times M$  grid
4      for each cell in the grid:
5          1. Get the density of white pixels
6          2. Get the centroid
7          3. Get the orientation
8      store these features in a vector
9  return featureVector
```

#### 3.4.5 Label Training Data

```
1  function labelTrainingData ()
2      for each letter:
3          label the letter on their corresponding y-axis
4          position
5  return labelArray
```

### *3.4.6 Classification Scheme*

```
1  function classification (featureVector, labelArray, newData,  
   labelData)  
2      create models using training set with the corresponding  
       group vector  
3      classify newData using SVM as one vs. all relation  
4  return classifiedData
```

## 4. Implementation

In this section, I will discuss the implementation of the specifications that were previously mentioned. More importantly, I will cover the main components of the system that I think should be given emphasis, and the unforeseen problems that I faced when developing the system, including proposals of solution to these problems. The system is divided into five main components: pre-process, segmentation, post-process, feature detection, and classification, whilst the function that labels the sample data is a supporting function for the classification.

The implementation of the system is done using MATLAB. This is because of the useful libraries that MATLAB has and how some of these libraries are capable of solving sub-tasks that the system had come across. Essentially, MATLAB provides an environment that I could adopt, i.e. “why code something that had already been done”; hence I will use the built-in functions for an effective implementation.

Before reading on, some terminology has to be established. During the rest of the report, I will be mentioning regions. A region could represent either a characters or noise and this region has information about itself, which includes the area, centroid and the bounding box.

### 4.1 Pre-process

The importance of this component is high, since this component will convert and prepare the image. Without the necessary pre-processing to the image, various problems will appear that may distort the results of the classification.

The significant parts of this function is converting the image from RGB, to grayscale then binary, and applying a median filter to reduce the noise present in the image. MATLAB has built-in functions which does these procedures that will save some implementation time. Furthermore, when using the median filter <sup>[24]</sup>, there is not much difference to where you position it into the code, but my system’s median filter had been implemented after converting the image to grayscale, as seen in Figure 7.

```
1 %convert image into grayscale
2 grayImage = rgb2gray(inputImage);
3 %apply median filter to reduce noise
4 grayImage = medfilt2(grayImage, [2 2]);
```

*Figure 7: Median filter implemented after image had been converted as grayscale.*

When applying the median filter, I had done so with a 2x2 kernel. This may differ depending on the image, but ideally, the kernel should be kept small in order to minimise information lost from the characters. I had kept the kernel small because the sizes of the characters in the sample data are small, but if the kernel is large then the median filter may erase the characters.

Once the image is converted into binary, the image needs to be inverted, i.e. black to white and vice versa, where the characters should be white and the background is black.



This is necessary because on the next component of the system, the regions can only be detected if they are represented as white pixels. A simple algorithm is to loop through the image, pixel by pixel, and using if statements to change the colour of the current pixel, as shown in the image below.

```
1  %invert colour
2  for i = 1 : size(binaryImage, 1)
3      for j = 1 : size(binaryImage, 2)
4          if binaryImage(i, j) == 0
5              binaryImage(i, j) = 256;
6          else
7              binaryImage(i, j) = 0;
8          end
9      end
10 end
```

*Figure 8: Algorithm that inverts the colour of the pixels.*

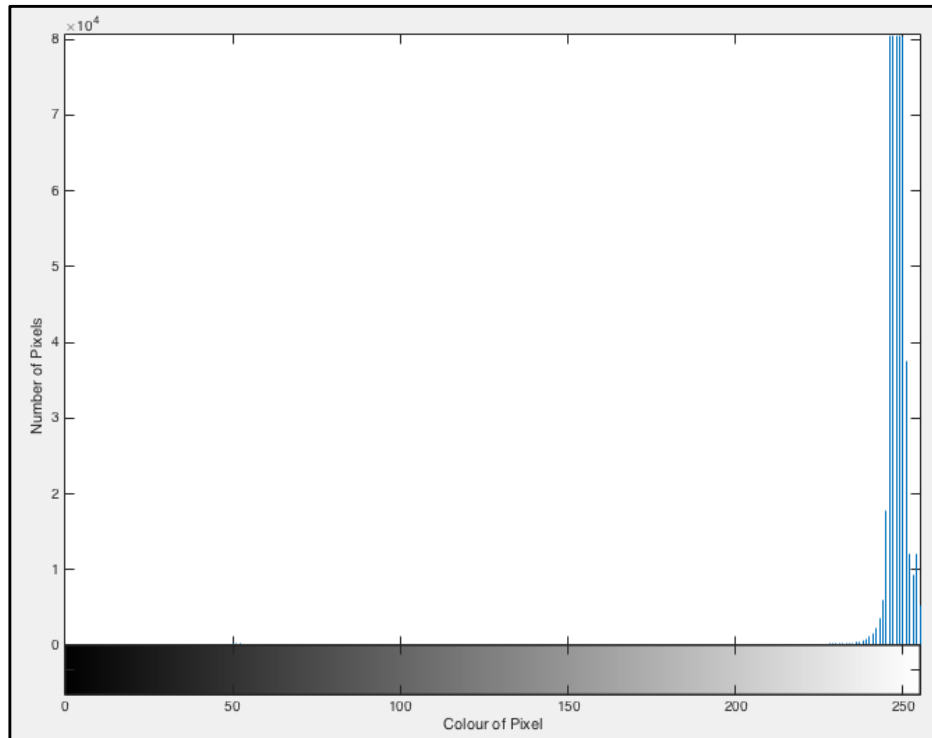
#### **4.1.1 Problem – Noise**

One of the most common problems in computer vision application is noise [25]. My application encounters the same problem, and noise may vary from image to image, which means the parameters of the noise reduction techniques would vary depending on how the noise is presented in the image. There is a fine line where the application must try to remove as much noise as possible, whilst keeping the detail of the characters. This trade-off is settled depending on the user's preference. The system utilises MATLAB's built in functions, such as a median filter, and bwmorph, in order to reduce the noise, but also affect the details of the characters.

#### **4.1.2 Problem – Threshold**

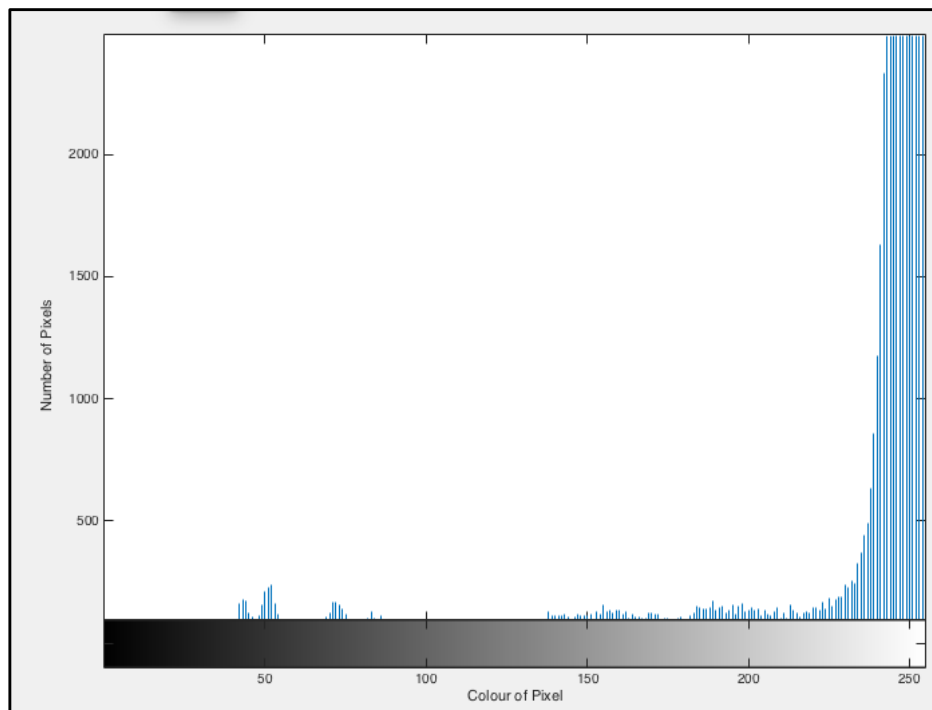
During implementation of the pre-processing stage, I encountered a problem, whereby the threshold used to convert the image from grayscale to binary varies [26]. Depending on the image and the handwritten characters, the threshold will vary based on these aspects. Since images vary depending on certain conditions such as the environment's lighting condition, the ideal solution is to make an adaptive threshold.

Initially I had planned to do this by creating an algorithm that performs an analysis of the histogram of the image [27]. The idea of the algorithm is to find the appropriate 'cut off point', where anything above it becomes a white pixel, whilst anything equivalent or less than is considered as an object. The object could either be the handwritten character or noise. However, this problem was not a priority for the system, where it could be manually solved with constraints and trial and error, but it must be adhered that this problem is present in the system.



*Figure 9: Histogram of 002.png (See Appendix C).*

Based on the histogram, Figure 9 indicates that the majority of the pixels are close to being white pixels, and these pixels mainly represent the background of the image. On the other hand, there are some gray pixels that are barely visible, as seen in Figure 10, which represents the regions of handwritten characters or noise.



*Figure 10: Closer look into the gray pixels.*

The 'cut off point' would usually lie in the range of 200-240, based on my sample data, whilst images taken from the camera usually have a 'cut off point' in the range of 150-175, where the image taken from the camera consists of decent, well light environment.

## 4.2 Segmentation

In this function, MATLAB does most of the work to obtain the regions (characters). To find the regions, I had used an 8-way connectivity to obtain a list of all of the regions, and using it to obtain the area, centroid, and bounding box of each region, as seen in Figure 11. The idea of 8-way connectivity is to obtain a list of all of the regions which are connected together by neighbouring pixels.

```
1 %find regions using 8-directed connected component
2 connectivity = bwconncomp(inputImage);
3 regions = regionprops(connectivity, 'Area', 'BoundingBox',
    'Centroid');
```

Figure 11: Using connectivity algorithm and regionprops to obtain information about objects in the image.

Although the regions contain the characters, it also contains other things such as remnants of noise. In this function, I had developed an algorithm which iterates through the 'regions', which is a struct that contains the three data types. Using the area, I had used an if statement that checks if the regions are within a specified range.

```
1 %apply a bounding box for each segmented letters to visualise the
  correctness of algorithm
2 for x = 1 : length(regions)
3     %ignores regions that have an area which are not within the
      specified range
4     if regions(x).Area < 55 || regions(x).Area > 1500
5         box = regions(x).BoundingBox;
6         rectangle('Position', [box(1), box(2), box(3),
          box(4)], 'EdgeColor', 'r', 'LineWidth', 1);
7         regions(x).Area = 0;
8         regions(x).BoundingBox = 0;
9         regions(x).Centroid = 0;
10    else
11        box = regions(x).BoundingBox;
12        rectangle('Position', [box(1), box(2), box(3), box(4)],
          'EdgeColor', 'g', 'LineWidth', 1);
13    end
14 end
```

Figure 12: Modifying the unwanted regions saved by regionprops.

For example, if the area of a region is not within the specified parameters, then change the value of the area, centroid, and bounding box to zero. If the region is within the range, then draw a bounding box to that region, which is used to visually see that the algorithm obtains the correct regions that represent the handwritten characters.

## 4.3 Post-process

The idea of this component is to fix the regions that are fragmented, but are not supposed to be [28]. For this, I had developed an algorithm which creates a new region

that contains both of the regions and deletes the previous, associated regions. This problem need to be fixed because both of the regions would be classified as individuals instead of as together, which could make the entire process of the classification incorrect. The reason for this problem varies, e.g. the character is split due to the user's handwriting or perhaps the threshold used are some of the reasons of losing connection of characters.

#### 4.3.1 Problem – Fragmented Characters

In this problem, due to the way characters are written and parts of the pre-processing stage, some characters split into more than one parts. Figure 13 shows the character 'I', split into two regions instead of just one.

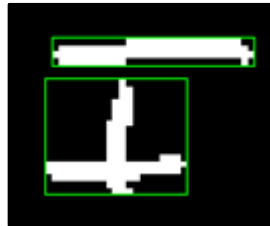


Figure 13: Example of fragmented handwritten character.

On the contrary, this problem may occur on normal circumstances, such as a 'j' or an 'i' would be considered as fragmented characters. The function in MATLAB recognises these regions as two instead of one.

Part of my solution to this problem is that I had implemented some MATLAB functions which dilate the image, so that the character's size expands as seen in Figure 14. Although this may be a good solution, if the image still contains some noise, then these will be amplified along with the characters, and the parameters that comes with the dilate function must be precise that it does not merge different regions that are not supposed to, such as a 'j' would merge into one region where it does not have the point.

```
1 %expand the size of the letter
2 se = strel('pair', [4, 1]);
3 binaryImage = imdilate(binaryImage, se);
```



Figure 14: Algorithm which dilates the image and the result from dilating the character 'I'.

The other part of my solution is the idea to create an algorithm that would locate and identify these fragmented characters and instead of having two regions, the algorithm would only produce one region. To do this, I had implemented an algorithm that determines the distance, both width and height wise, of regions, and depending on the distance set by parameters, and then I would get the coordinates that would contain both regions as one region, as seen in Figure 16.



Figure 15: Before (with dilate)



Figure 16: After (with dilate and algorithm).

#### 4.4 Feature Detection

Before extracting features from the character, I had to implement an algorithm which visually creates a grid for each character. This is achieved by the following code:

```
1 for y = boundingBox(2) : (boundingBox(4) / 3) : boundingBox(2) +
  boundingBox(4) - 1
2   for x = boundingBox(1) : (boundingBox(3) / 8) : boundingBox(1)
     + boundingBox(3) - 1
```

The part of the code that has been highlighted in yellow represents the dimensions of the grid, where in this case, it divides each of the characters into an 8x3 grid. As you can see, the dimensions of the grid are easily changeable, where you would just manually increase or decrease the amount of cells you wish to have. Ideally, the dimensions of the grid should be relatively neither too small nor too large because this would make it difficult to distinguish the characters. This would visually produce the following:

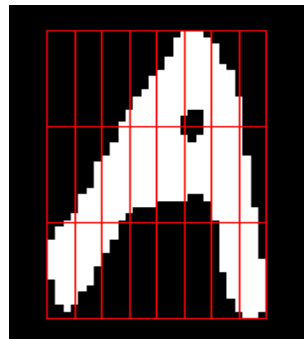


Figure 17: The character 'A' divided into cells to represent a grid.

Once I had achieved the above, I had to develop an algorithm which will iterate through each of the pixels in each cell of the grid. A cell is the subsections of the grid, where Figure 17 had 24 cells. These cells contain the features.

- To get the density of white pixel of a cell, I had implemented a counter for the amount of white pixels that are within a cell, and divide it by the total number of pixels within the cell.
- To get the average distance of a cell to the centroid, I had accumulated the x and y values from all of the white pixels that are present in the cell, then divide them by the number of white pixels in the cell.
- To get the orientation of the cell, I had created a temporary matrix which is a replica of the cell, then applied regionprops to obtain the centroid of that cell.

These features are stored as a vector, which consists of all of the features in the order of orientation, distance and density for each cell. Each feature vector represents a character. As a result, I was able to achieve 96 features from a single character.

## 4.5 Classification

The concept of a SVM is a binary classification scheme. Since my sample data consists of more than two classes, I needed an unconventional way to classify all of my classes using SVM. The following is one of the many ways of solving the problem. Other solutions may include using neural networks instead of SVM.

For this component of my system, I had used Cody Neuburger <sup>[29]</sup>, which builds models from the training set and their associated vector, and classifies the test set using the SVM classifier as a “one vs. all relation”, as seen in Figure 18, lines 2-7.

To build the models, Neuburger’s code iterates through each unique classes, where the current class is trained against all of the other classes. These models are then used to classify the test set by using `svmclassify`, as seen in Figure 18, lines 9-16.

```
1  %build models
2  for k=1:numClasses
3      %Vectorised statement that binarizes Group
4      %where 1 is the current class and 0 is all other classes
5      GlvAll=(GroupTrain==u(k));
6      models(k) = svmtrain(TrainingSet,GlvAll);
7  end

8  %classify test cases
9  for j=1:size(TestSet,1)
10     for k=1:numClasses
11         if(svmclassify(models(k),TestSet(j,:)))
12             break;
13         end
14     end
15     result(j) = k;
16 end
```

*Figure 18: Neuburger’s multi-class SVM.*

The function returns an array of numbers that are between 1-26. My function converts these numbers into their corresponding character, as seen on the figure below.

```
1  characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
2  classified = '';
3  for i = 1 : length(svmStruct)
4      classified = strcat(classified, characters(svmStruct(i)));
5  end
6  disp(classified);
```

*Figure 19: Displaying the classified image.*

If the test set is used against the training set, then a percentage is output, seen in Figure 20, which represents the accuracy of the system based on the features and classification used. Otherwise the function will just display the classified results.

```

1 correctness = 0;
2 for i = 1 : length(labelData2)
3     if characters(svmStruct(i)) == labelData2(i)
4         correctness = correctness + 1;
5     end
6 end
7 disp('Accuracy of Classification Scheme:');
8 disp(double(correctness / length(svmStruct)));

```

Figure 20: Display the accuracy against the test data.

## 4.6 Labelling Sample Data

### 4.6.1 Problem – Label Training Data

When using MATLAB's built in function, `regionprops`, it generates a struct containing what you have specified it to have. In my case, I had used `regionprops` to get the area, bounding box, and centroid. `Regionprops` goes through the entire image from the x-axis then y-axis, for example,  $x = 1, y = 1 \dots n$ , then  $x = 2, y = 1 \dots n$ .

Due to the way `regionprops` functions, it is difficult to label the characters because they do not simply go from A-Z. For example, Figure 21 would give the values in the struct as 'D', 'B', 'A', then 'C'. This is because characters that have a low x-axis would be detected first. As a result, I cannot just label each of these characters based on the position of which they are stored in the struct.

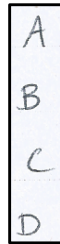


Figure 21: Snippet of the sample data.

Instead of manually labelling each character in the training images, my solution to this problem is to develop an algorithm which checks the position of character. I was able to successfully implement this algorithm due to how I had structured my training and test sets. Each character in the sample data is written within a grid, where each cell contains a character. Knowing this information, I can estimate a range of the position of each character based on its y-axis. The following image is a snippet of code from the algorithm, which applies a character to a string depending on the value of the y-axis of the character.

```

1 for i = 1 : length(regions)
2     if regions(i).Area ~= 0
3         boundingBox = regions(i).BoundingBox;
4         %disp(boundingBox);
5         if boundingBox(2) >= 210 && boundingBox(2) <= 260
6             labelData = strcat(labelData, 'A');
7         elseif boundingBox(2) >= 270 && boundingBox(2) <= 330
8             labelData = strcat(labelData, 'B');
9         elseif boundingBox(2) >= 350 && boundingBox(2) <= 410
10            labelData = strcat(labelData, 'C');

```

Figure 22: Code snippet of the algorithm to label the sample data.

## 5. Unit Testing

The idea of these test procedures is to ensure that the components of the system work as intended. The section is subdivided into the main stages of the system, which are: pre-process, segmentation, post-processing, feature detection, label training data and classification.

### 5.1 Pre-process

#### 5.1.1 Filtering Noise

In order to filter noise from the images, I had used a combination of techniques, which are: median filter, converting grayscale to binary using an appropriate threshold, making the outside border black, and setting isolated noise (surrounded by black pixels) as background. Figures 23 and 24 shows a 'before and after' applying the noise filtering techniques. Despite starting with an image covered by masses of noise, by using a few modifications of noise filtering techniques could eliminate most of them.

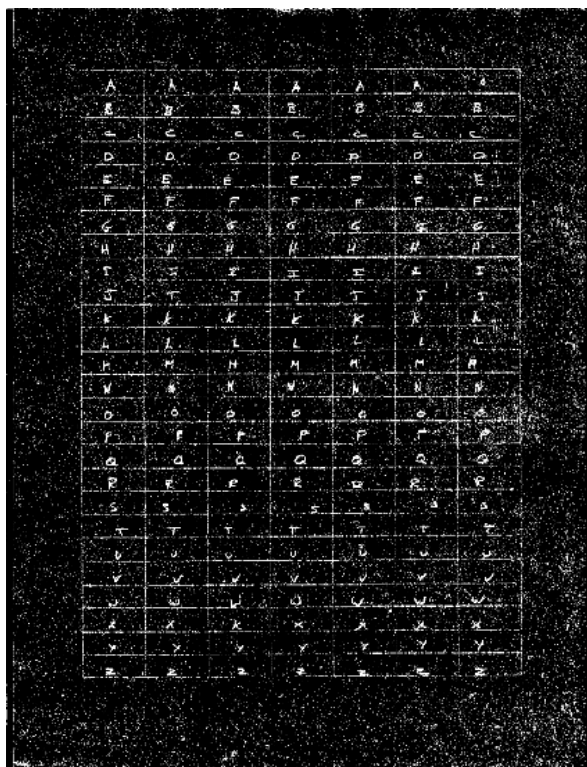


Figure 23: Image before filtering.

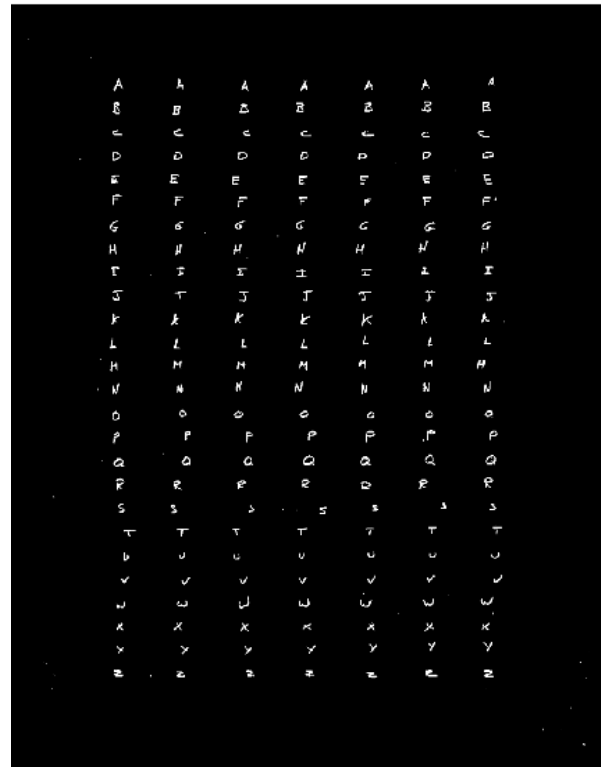


Figure 24: Image after filtering.

When applying just the median filter, the system is capable of achieve a reasonably filtered image, as seen in Figure 25. However, the noise needs to be further reduced, since having less noise would enable to achieve better results further into the system, i.e. less records in the struct meaning less iteration, which improves the performance in terms of time complexity. The median filter does not work well with the noise that are closely bundled together because the median filter uses a neighbourhood that comes with a mathematical formula to obtain the median of neighbourhood and if the



neighbourhood consists of mostly white pixels then the output would be a white pixel [24].

On the contrary, when applying the function of morphological operation of majority, most of the noise is removed, as seen in Figure 26. This function is able to achieve the results due to the how the noise are pixels that are separated enough, rather than joint into a big blob. For example, if a small cluster of white pixels are surrounded by black pixels, then the group of white pixels are converted into black pixels [16].

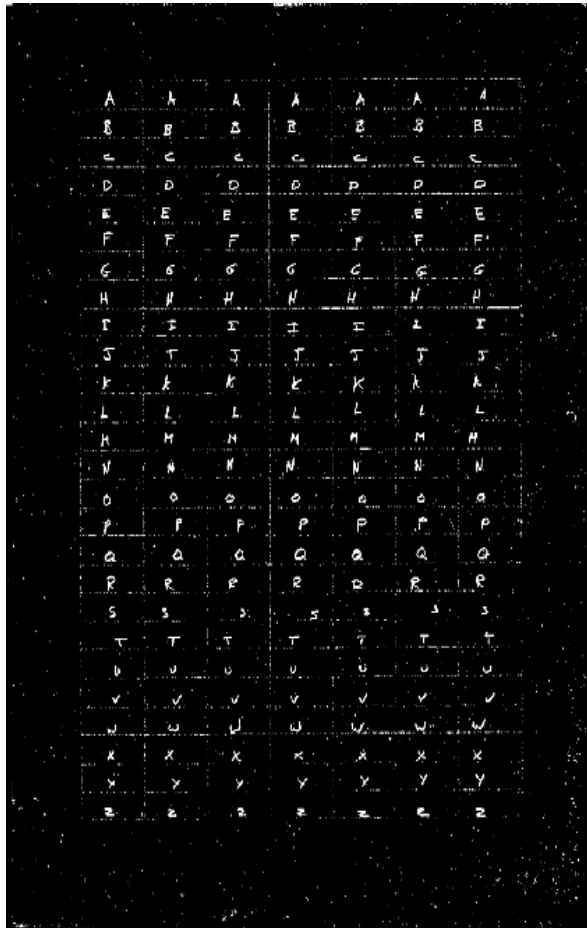


Figure 25: Image with median filter.

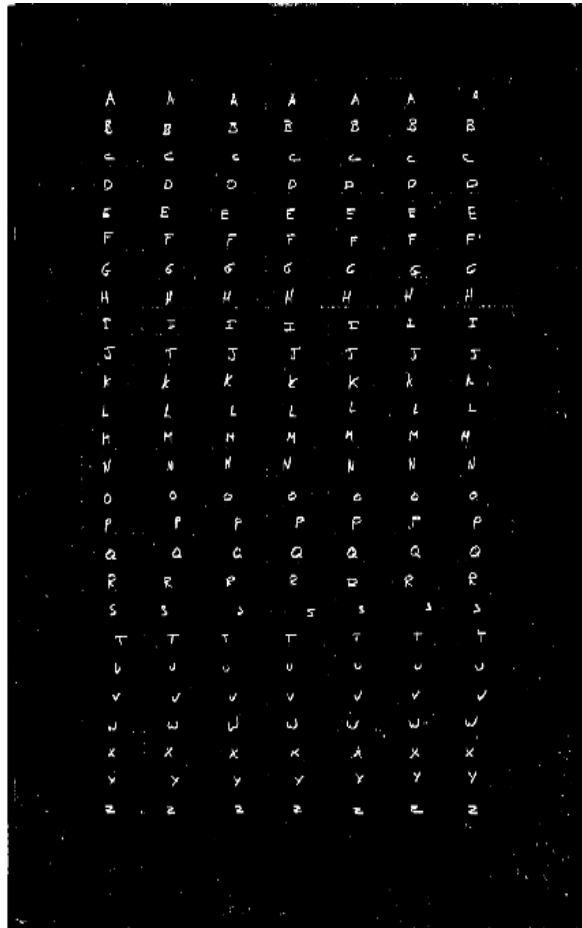


Figure 26: Image with bwmorph (majority).

Between these two techniques, the morphological operation has the upper hand in noise reduction. However, in terms of keeping information of the characters, the median filter is superior as the morphological operation has the tendency of removing information from the characters due to the way it works. For example, if a character is very fragmented, then the fragments may turn into background. This would ruin the end results because the characters are not fully representing what they should appear.

Furthermore, since I had scanned my training and test sets, the part of the borders of the scanned image consists of black because the A4 does not completely fit the scanner. I could have manually cropped them out, but MATLAB has an operation called 'imclearborder', which does this operation [30]. The Figures 27 and 28 shows a before and after images when applying the operation. The changes are not noticeable at first glance, but this operation is useful when combined with the other techniques because clusters of noise that are touching the border of the image cannot be removed using the other techniques.

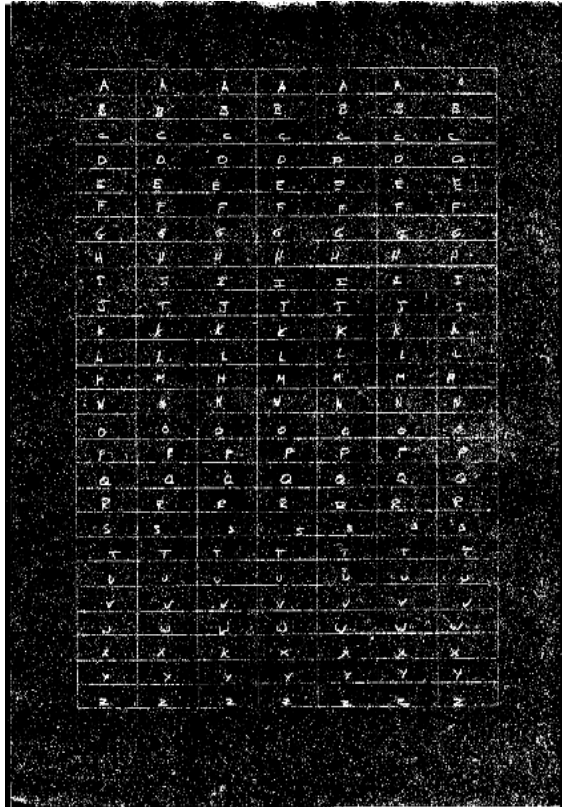


Figure 27: Image before border cleaning.

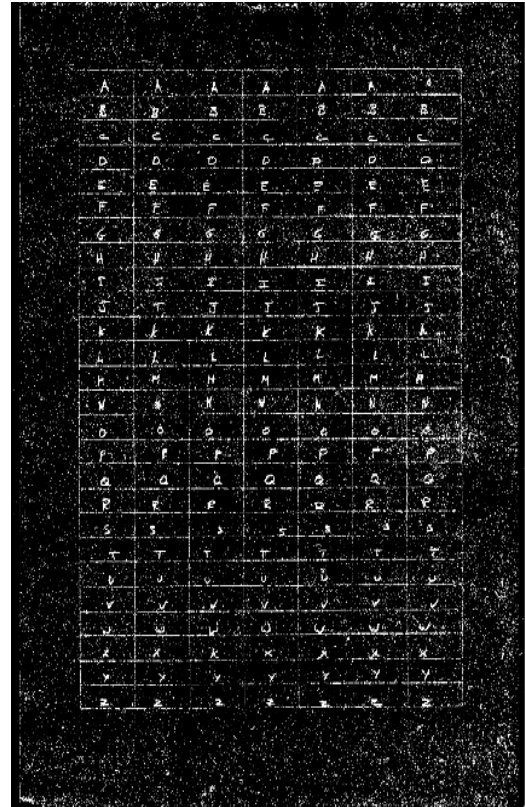


Figure 28: Image after border cleaning.

Finally, when it comes to selecting the threshold to convert the image from grayscale into binary, plays a big part in terms of noise filtering, but also a delicate parameter that should be considered based on the image. Using the right threshold can achieve better results compared to the median filter or the morphological operation of majority, but using the incorrect threshold could result in more noise and/or losing information of the characters.

The threshold used for the images above, from Figure 23-29, was set to 245/256, but the image below, Figure 30-32, had used a threshold set to 238/256. The results show that the majority of the noise had been removed, but the some of the characters are affected, where pixels that are part of the characters had been deleted. Additionally, it is also noticeable that the character has a clearer shape compared to beforehand, but information is lost in the form of the black pixels that appears inside the character.

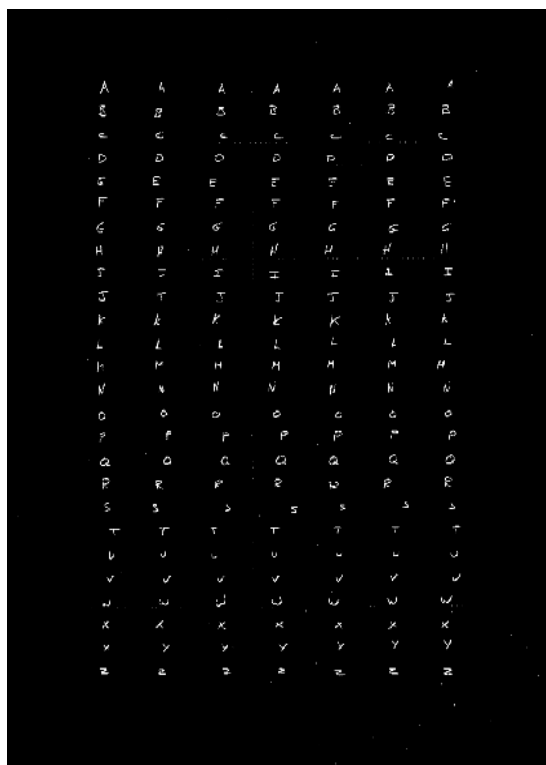


Figure 31: Image with 238/256 threshold.



Figure 30: Example of character after applying 245/256



Figure 29: Example of character after applying 238/256 threshold.

To solve the problem that had been mentioned above, where information is lost when using a reasonable threshold is to use the other noise reduction techniques, such as median filter and morphological operation of majority. The image below, Figure 32, shows the results after applying the techniques. As you can see, the character is able to retain some of the information to a certain degree, hence the results 'imitate' the original image.

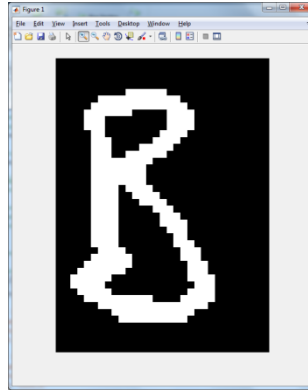


Figure 32: Example of character after all noise reduction techniques.

## 5.2 Segmentation

In order to test that MATLAB's function `regionprops` is working as it should, I had temporarily implemented an algorithm, which is a simple counter that accumulates the number of regions that are within the image. I had implemented the algorithm inside the post-processing function after the algorithm had finished in order to obtain the correct number of characters in the image, i.e. the algorithm had fixed the fragmented characters. Using one of the training set, the system correctly obtains the correct number of regions, which is 182 characters.

```

1 counter = 0;
2 for x = 1 : length(regions)
3     if regions(x).Area ~= 0
4         counter = counter + 1;
5     end
6 end
7 disp(counter)

```

```

>> main
Warning: Image is too big to fit on screen: displaying at 25%
> In images.internal.initSize (line 71)
   In imshow (line 305)
   In main (line 9)
    182

```

Figure 33: Accumulate the number of characters in one sample set and displaying the results.

Furthermore, another way to check if the regions obtained are correct, I had drawn the smallest bounding box containing the region. The green bounding boxes represents the characters, whilst the speckles of red bounding boxes are remnants of noise as shown in Figure 34.

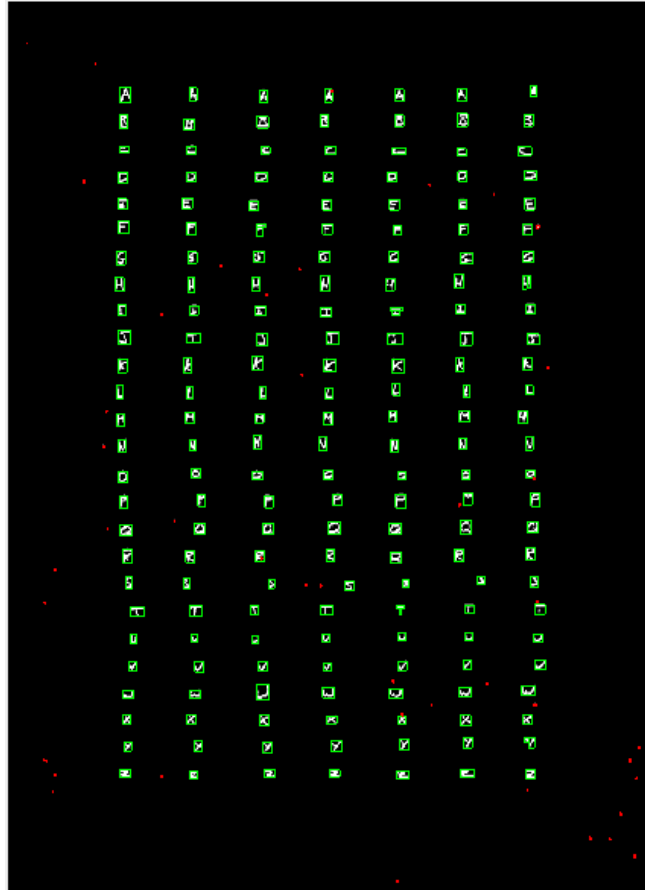


Figure 34: Displaying the detected regions.

### 5.3 Post-process

To check if the algorithm of combining fragmented regions, I had used several test samples gathered from the sample data. The Figure 35 shows a scenario where a character is split into two, width wise. Whilst Figure 36 shows how the character was merge into one regions, where before there were two values in the struct to represent both regions, whereas Figure 37 changes the values of the record of the regions, then creates a new record to represent the whole character as one. Hence solving this problem by merging the regions as one.

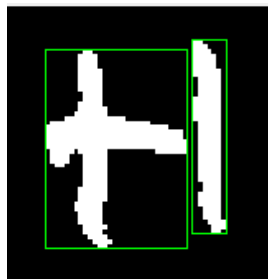


Figure 35: Example of a fragmented character, width wise.



Figure 36: Example of a fragmented character fixed.

Figure 37 shows two screenshots of the 'Variables - regions' window. The left screenshot shows a '2x1 struct with 3 fields' table with 2 rows of data. The right screenshot shows a '3x1 struct with 3 fields' table with 3 rows of data, where the fragmented regions from the left have been merged into a single region.

Fields	Area	Centroid	BoundingBox
1	370	[20.3216,29.5595]	[8.5000,9.5000,30,42]
2	195	[42.8564,27.3538]	[39.5000,7.5000,7,41]

Fields	Area	Centroid	BoundingBox
1	0	0	0
2	0	0	0
3	300	[31.5890,28.4567]	[8.5000,7.5000,38,44]

Figure 37: Before and after fixing the regions.

Another test for this problem is when the fragmented character is separated height wise. Similarly, the results prove that the algorithm works, where it creates a new record, in order to fully represent the character as one region instead of two.



Figure 38: Example of fragmented character, height wise.



Figure 39: Example of fragmented character fixed.

Figure 40 shows two screenshots of the 'Variables - regions' window. The left screenshot shows a '2x1 struct with 3 fields' table with 2 rows of data. The right screenshot shows a '3x1 struct with 3 fields' table with 3 rows of data, where the fragmented regions from the left have been merged into a single region.

Fields	Area	Centroid	BoundingBox
1	416	[35.9279,20.1274]	[13.5000,10.5000,44,36]
2	244	[41.8320,52.9344]	[21.5000,48.5000,40,9]

Fields	Area	Centroid	BoundingBox
1	0	0	0
2	0	0	0
3	300	[38.8799,36.5309]	[13.5000,10.5000,48,47]

Figure 40: Before and after fixing the regions.

On the contrary, the centroid on the new record should be different to the previous centroids of the regions, which means a new centroid should be created. My method of obtaining the new centroid is to take both centroids from the two previous regions and get the average. For example:

$$\begin{aligned} newCentroid[x] &= \frac{(region1Centroid[x] + region2Centroid[x])}{2} \\ newCentroid[y] &= \frac{(region1Centroid[y] + region2Centroid[y])}{2} \end{aligned}$$

Using the regions from Figure 38 as an example, I had calculated the new centroid for the region by:

$$\begin{aligned} region1Centroid[x,y] &= [35.9279, 20.1274] \\ region2Centroid[x,y] &= [41.8320, 52.9344] \end{aligned}$$

$$newCentroid[x] = \left[ \frac{35.9279 + 41.8320}{2} \right]$$

$$newCentroid[y] = \left[ \frac{20.1274 + 52.9344}{2} \right]$$

$$newCentroid[x,y] = [38.87995, 36.5309]$$

This is where  $newCentroid[x,y]$  is the new centroid for the character.

This test proves that the algorithm does obtain the correct centroid as intended when compared to the value in Figure 40. As for the area of the new record, the value of it does not matter at this stage of the system, as long as it is not set to zero since the area property was mainly used in the segmentation stage to get rid of unwanted regions, but we need this new region and not the previous regions.

## 5.4 Feature Detection

When extracting the features from each cell, my algorithm does so by iterating from the top left corner up to the bottom right corner, as demonstrated in the table below. The dimensions do not fully represent the end system, since the dimensions used in the system is a 8x3, but it is give you the idea of how the algorithm iterations through each cells in the grid. This is useful when it comes to looking at the feature vectors to determine their correctness.

1	2	3
4	5	6
7	8	9

Figure 41: Example of how algorithm traverses from cell to cell.

To check the correctness of the algorithms, I will be using the image below as my example. From the image below, Figure 42, we can see that some of the cells of the character consists of all black pixels, whilst some are filled with white pixels. Based on this information, I know which cells should have values of zero, and which cells should have values greater than zero.

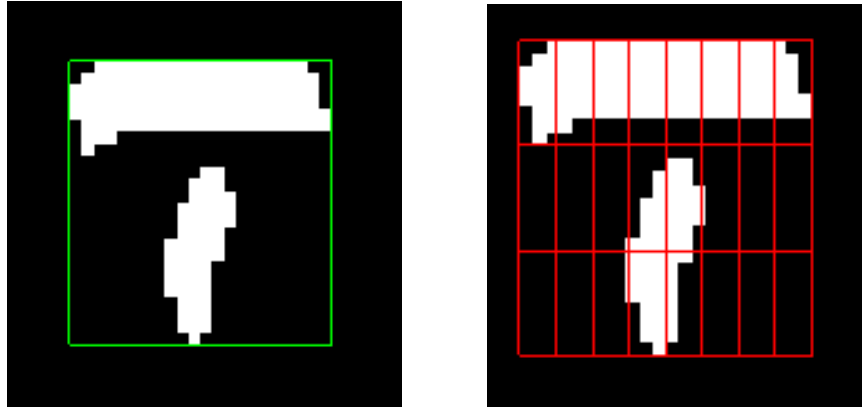


Figure 42: Visual example of the algorithm applying a grid into the character.

The next three tests are to determine if algorithm is obtaining the features correctly. I had done so by making individual test for each features using Figure 43 as my example.

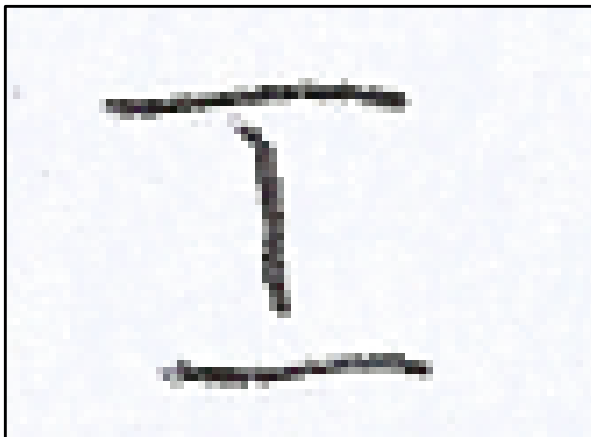


Figure 43: Original image.

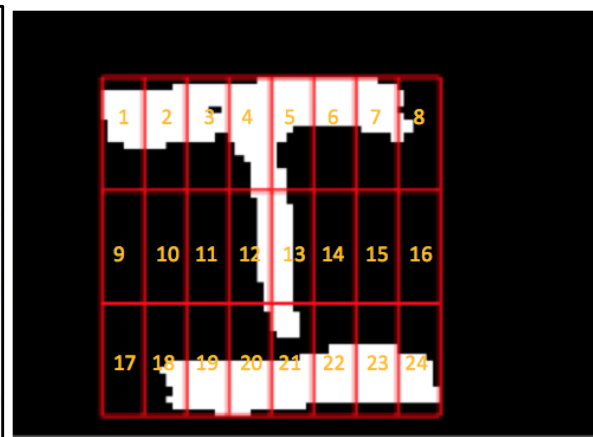


Figure 44: Visual grid with labelled cells.

### 5.4.1 Density

Upon running the algorithm, I obtain a feature vector which can be seen in the Appendix B – Density Vector. When comparing the feature vector towards the Figure 44, according to my feature vector, cells 9, 10, 11, 14, 15, 16, and 17 have no density. In other words, these cells do not have any white pixels, which is the expected outcome.

Figure 45 shows the number of white pixels and the number of pixels in the first cell. From this I can work out the density of the cell by:

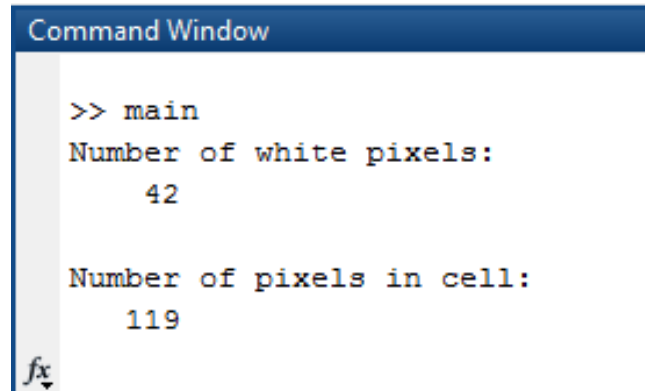
$$\text{Density} = \frac{\text{Number of white pixels}}{\text{Number of pixels in cell}}$$



For example:

$$0.3529 = \frac{42}{119}$$

As you can see, this value matches the value to the first value in the feature vector, and this proves that the algorithm is computing the density correctly. Note that the algorithm includes the bounding box surrounding the character, but are processed as both black or white pixels, and not the colour of the bounding box.



```

Command Window

>> main
Number of white pixels:
    42

Number of pixels in cell:
    119
fx

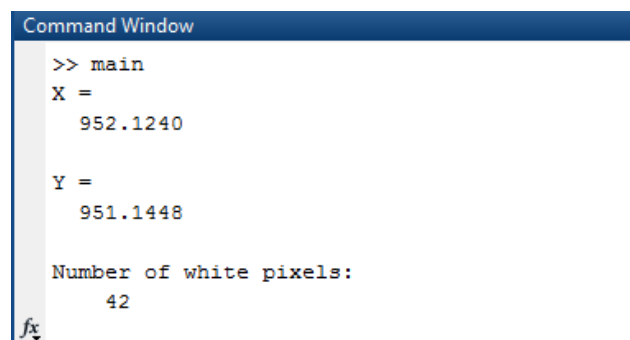
```

Figure 45: Displays the information of cell number 1 in terms of density.

#### 5.4.2 Centroid

Similarly, using the algorithm to calculate the average distance to the centroid, I had obtained the feature vector, seen in the Appendix B – Centroid Vector.

Notice that the number of features points doubled compared to the previous feature vector. This is because the features in this vector consists of the x and y axis of the average centroid of each cells. For example, cell 1 has an average distance to the centroid of [15.4927, 14.9183]. In order to check if this value is correct, I had calculated the accumulated x and y distance to the centroid, and the number of white pixels within the first cell, which gives me the following in Figure 46.



```

Command Window

>> main
X =
    952.1240

Y =
    951.1448

Number of white pixels:
    42
fx

```

Figure 46: Displays the information of cell number 1 in terms of centroid.

By calculated the following:

$$\text{Average distance of white pixels in cell from centroid} = \frac{\text{region1Centroid}[x, y]}{\text{Number of white pixels}}$$

I had used this to check that my values are correct in the feature vector. For example,

$$x = \frac{952.1240}{42} \quad y = \frac{951.1448}{42}$$

$$\text{Hence: } [x, y] = [22.67, 22.65]$$

This is the same value as the first two values from the feature vector. This concludes that the algorithm is working as intended. Furthermore, some of the values in the feature vector are zeros, which is what to be expected since these are the cells which have no white pixels in them, meaning the algorithm cannot compute this feature.

### 5.4.3 Orientation

For the algorithm to calculate the orientation of each cell, I had obtained the feature vector which can be seen in Appendix B – Orientation Vector.

In order to test that the algorithm works, I need to check that the correct part of the image is copied into a matrix. Figure 47 shows the matrix which represents the top left corner of the character in cells that was obtained from the algorithm. From this, I had applied MATLAB's function, `regionprops`, which calculates the orientation of the image for me.



*Figure 47: Cell number 1 of the character in Figure 44.*

## 5.5 Labelling Training Data

In order to test the algorithm that I had used to label the training or test sets, I had edited one of the training sample, as seen in Figure 48, where I had left just the row of the character 'A' in the image. The results show that the algorithm is able to detect that there row of characters are in fact the character 'A', as shown in Figure 49.



Figure 48: Edited sample data.

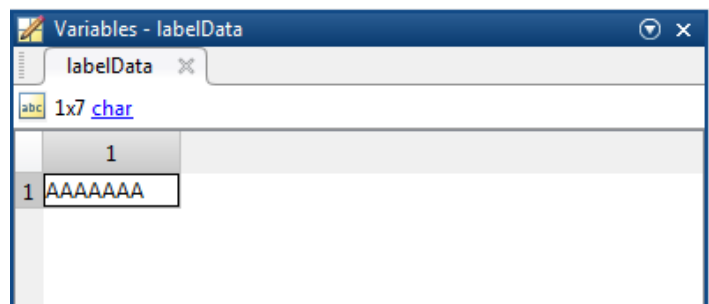


Figure 49: Results displaying as expected.

Another test which I had performed is checking a column of characters from the image, as seen in Figure 50, where the output should give me all of the characters in the alphabet. The output gives the results in Figure 51. To clarify that Figure 51 is all of the alphabets, I manually applied a simple sort to the variable, giving me the results shown in Figure 52. As you can see, the algorithm is working as intended.

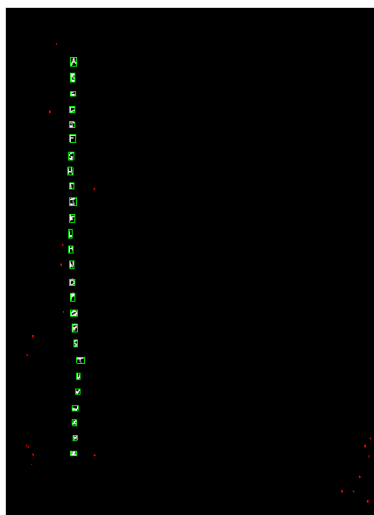


Figure 50: Edited sample data.

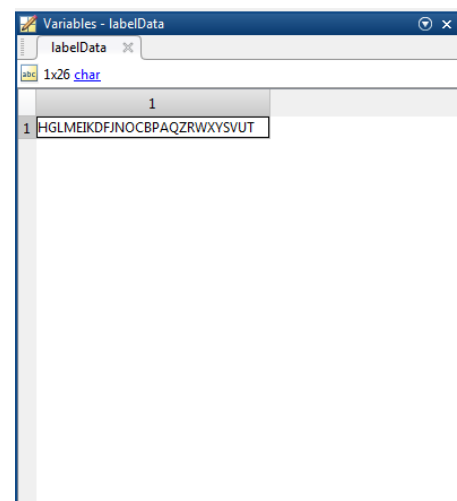
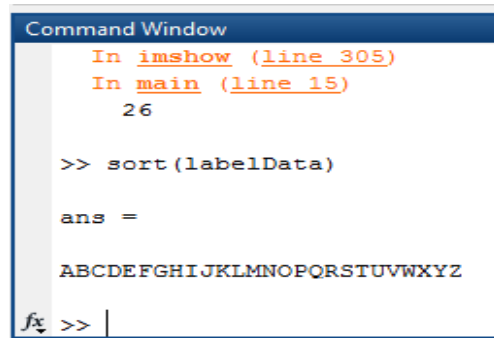


Figure 51: Results from algorithm.



```

Command Window
In imshow (line 305)
In main (line 15)
26

>> sort(labelData)

ans =

ABCDEFGHIJKLMNOPQRSTUVWXYZ
fx >>

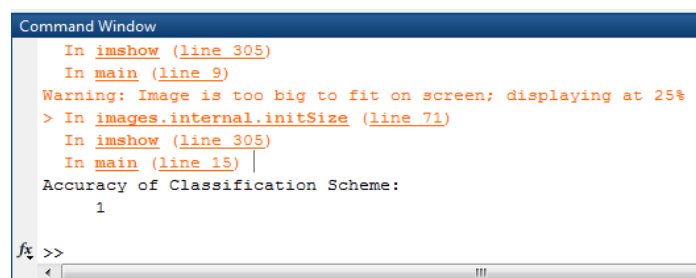
```

Figure 52: Sorted results.

## 5.6 Classification

*Note: for the classification tests, I had used the images which can be seen in the Appendix C [001.png and 002.png].*

One way of testing that my classification scheme is working as intended is to use the same image, 001.png as my training and test set. The output must be in the range of 0-1, where the 0 has no correct matches, whilst 1 has classified all of the characters in the test set correctly. The results shown in Figure 53 outputs a 1, which means that the classification scheme successfully classified all of the characters in the image using the training image as expected.



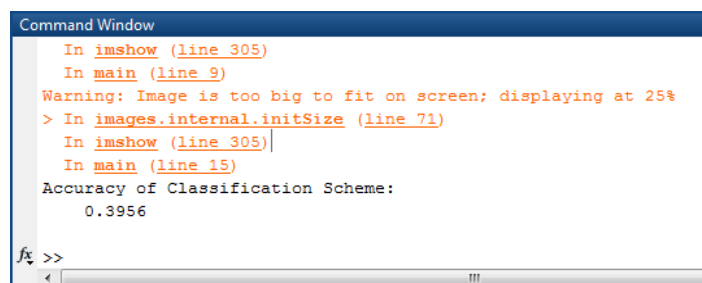
```

Command Window
In imshow (line 305)
In main (line 9)
Warning: Image is too big to fit on screen; displaying at 25%
> In images.internal.initSize (line 71)
In imshow (line 305)
In main (line 15)
Accuracy of Classification Scheme:
1
fx >>

```

Figure 53: Results after classifying the image, similar to the training data.

For this next test, I had used a different image to be classified, where 001.png as my training and 002.png as my test data, which can be seen in the Appendix C. The results show as expected, where the accuracy of the classification scheme is less than one. The classification scheme gave a low accuracy score which is approximately correct since the training set lacks sample data in order to classify the test set more accurately.



```

Command Window
In imshow (line 305)
In main (line 9)
Warning: Image is too big to fit on screen; displaying at 25%
> In images.internal.initSize (line 71)
In imshow (line 305)
In main (line 15)
Accuracy of Classification Scheme:
0.3956
fx >>

```

Figure 54: Classifying image 001.png against 002.png.

## 6. Evaluation & Results

In this section, I will conduct further test on my system, where I will highlight the pros and cons of the system. To be specific, I will be testing the algorithms that had been integrated into the system and analysing them in terms of effectiveness against the intended purpose of the actual algorithm.

Looking at stages of the system in terms of evaluation, I believe that the algorithms that I had provided are sound solutions, where they are capable of achieving of what they were intended to do. However, the only concerns with these algorithms are the parameters that come along with them. This is because the values of these parameters are entirely dependent on the image. For example, when taking an image, there are a number of factors which could affect the quality of the image. In particular, different lighting conditions could cause the system to fail due to the threshold used to convert the image from grayscale to binary.

More specifically, the threshold used for the sample data is:  $238/256$ , which is specialised and tweaked to the sample data by trial and error, but if I were to take an image from my mobile-phone, the appropriate threshold would be in the range of  $150/256$ , assuming that there is decent lighting conditions in the area. Figure 56 shows the original image taken from a mobile-phone's camera that had been processed using a threshold of  $238/256$ , whilst Figure 57 shows the image processed using a threshold of  $150/256$ . Using my threshold,  $238/256$ , the threshold used is too high which meant that it is converting the characters as black pixels instead of white pixels.



Figure 55: Original image.



Figure 56: Image after  $238/256$  threshold.

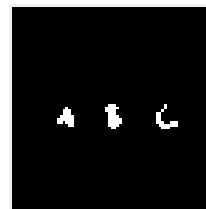


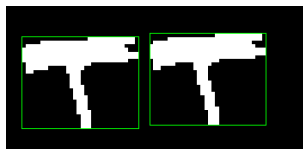
Figure 57: Image after  $150/256$  threshold.

At this point in time, the system would benefit from an algorithm which has an adaptive threshold, but an algorithm of such requires a multitude of testing, which means time allocation just for the algorithm, and time which could be used for more important sections of the system. For now, the system is constrained to a specific condition, where if you wish to classify characters from paper, you would need to scan the paper first, since this is how I had gathered the image from the sample data, and the system should be able to convert the image into binary using the current threshold.

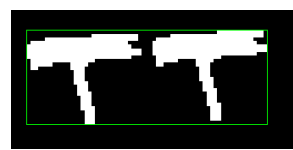
In the post-processing stage, the algorithm that joins separated regions has parameters which are distance measurements between regions. These parameters need to be considered carefully when handling an image where there are regions that are

positioned closely to each other, assuming that they are not supposed to be one region. For example, if the image contains characters which are closely written together, as seen in Figure 58, then the algorithm will join both characters together as one region instead of two. This is an important concern because in the real world context, characters tend to be handwritten closely to each other.

If the distance between the handwritten characters are far enough then the algorithm would not join the characters. As for the real world context problem, the parameters could be further experimented to see the appropriate values for them, but I had kept these values at a reasonable distance because these are the correct distance when applied to my sample data.



*Figure 58: Before post-processing stage.*



*Figure 59: After post-processing stage.*

Furthermore, I believe that this algorithm may address the fragmented characters problem, but there are other algorithms that may be more efficient and robust compared to my method. These algorithms will be further discussed in the future improvement section, but as an example of one type of algorithm is using the grayscale version of the image, check the intensity between regions and merge based on a similarity scale.

The algorithm that I had implemented to label my data is only applicable to my sample data. This is because the algorithm is hard-coded, where the characters will be labelled depending on the position they had been written in. I purposely had a grid for the characters, which meant that I knew the position of the characters, where the positions of the characters are based on the y-axis. This meant that a region is determined by a range, where each range would specify the corresponding character to that region.

From the perspective of performance, currently, the system is not running to its optimal potential. The main objective of the project is to ensure that the system is capable of performing the required aims and objectives, so the performance is considered as an additional section, where if given more time then that time could be allocated into improving the quality of coding.

Moreover, when breaking down the system based on its functions, obtaining the features would be considered the most performance intensive. For example, in order to gather the features that I required, the algorithm required nested loops, where a multitude of temporary variables are used to store values of features. The number of characters present in the training/test sets or the input image would also have an

influence to the performance of the system, where the performance would have a linear time complexity as the size of the sample data is increased.

In terms of the features that I had used, the most useful feature for the classification scheme is the density. When using just this feature, the system is able to achieve a decent accuracy rate, whilst the other two features had little influence in increasing the accuracy rate.

However, more importantly, the other two features do help in making the characters more unique and distinguishable, which mean that the more distinguishable the characters are, then the more likely a convergence could be made when training the features vectors into the SVM. I had discovered that with the other two features, the accuracy rate increases by 6%. These tests that I had conducted consist of using the training set and the input image which can be seen in the Appendix C [001.png and 004.png].

```

Command Window
> In images.internal.initSize (line 71)
  In imshow (line 305)
  In main (line 4)
Warning: Image is too big to fit on screen;
> In images.internal.initSize (line 71)
  In imshow (line 305)
  In main (line 9)
Accuracy of Classification Scheme:
    0.5769
fx >>

```

Figure 60: Without additional features.

```

Command Window
> In images.internal.initSize (line 71)
  In imshow (line 305)
  In main (line 4)
Warning: Image is too big to fit on screen;
> In images.internal.initSize (line 71)
  In imshow (line 305)
  In main (line 9)
Accuracy of Classification Scheme:
    0.6429
fx >>

```

Figure 61: With additional features.

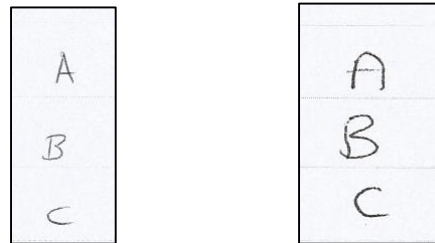
The following table is the accuracy and error rate of the entire training set against the test set, which can be found in Appendix C, where 000.png as my training set and 005.png as my test set. Note that 000.png consists of all of the sample data that I had except 005.png.

Accuracy Rate	Error Rate
72%	28%

Figure 62: Performance of classification scheme.

To check the accuracy of each individual features, I had tested them individually, where I had applied one feature at a time to the classification scheme. These tests resulted in the density being the most accurate because the other two features are unable to obtain a convergence using the SVM or have a high error rate. This means that the orientations and average distance from the centroid features are not unique enough to be able to identify the characters.

Due to the features that I had used, my system is highly dependent on the training set. Ideally, the training set should consists as many possible variations of the characters, which is difficult to achieve since there are individual handwriting that are unique based from person to person. Assuming that there are minor variations with the handwritten characters, such as shown in Figure 63, the system is more than capable of achieving an accuracy rate greater than 70%.



*Figure 63: Snippet from the sample data.*

Basically, if the shape of a character is not in the training set and is completely different, such as the character 'A', shown in the figure above, then the system would most likely fail to recognise and classify the character as an 'A', due the shape of it.

On a contrary, since I do not have a large sample size, I had conducted some tests to speculate whether there is a trend to the accuracy rate of the system's classification scheme when the number of training sample is increased. The following Figure 64 shows that there is some truth towards the speculation of the accuracy rate gradually increasing as the number of training data is also increased. However, since this is only a speculation, there is no guarantee that accuracy rate will increase when a larger sample size is used.

In addition, I had also discovered in the same test that my system's classification scheme is prone to model overfitting. This can be seen from the table where the accuracy rating decreases when the number of training set was at 2. I believe that the reason for this decrease is mainly because of the type of features I had used. If the test set contains characters which are completely different, in terms of their variations, such as size, shape, etc., compared to the characters in the training set, then the system would not be able to classify the characters in the test set correctly, i.e. the error rate would be high.



Number of training set (per page(s))	Number of test set (per page)	Accuracy Rate
1	1	64%
2	1	60%
3	1	62%
4	1	69%
5	1	72%

Figure 64: Test results with accumulating training samples.

Essentially, the system is unable to generalise when it comes to unknown data sets. Details which cover the solutions involving this problem are covered in the future improvement section. This problem was unforeseen and due to the deadlines meant that I was unable to react and adapt my approach to the problem.

In these following tests, I had experimented with the performance of the feature extraction techniques, which includes individual and combination of the features to see what accuracy rate they would provide. Note that these test use 001.png as the training set and 005.png as the test set seen in Appendix C.

Feature(s)	Accuracy Rate	Error Rate
Density	64%	36%
Orientation	25%	75%
Distance	No convergence	N/A
Density & Orientation	60%	40%
Density & Distance	65%	35%
Orientation & Distance	36%	64%
Density, Orientation & Distance	64%	36%

From this, I can evaluate that the density feature performs better than the orientation or distance features. The distance feature on its own was not able to find convergence, which means that a model cannot be made due to the SVM is unable to plot a hyperplane to separate the class against the rest of the classes. This means that the feature is not providing enough unique descriptors or feature points for the characters.

Furthermore, when applying a combination of features, the accuracy rate for density and orientation decreases the accuracy rate when compared to only using the density feature. This may indicate that using the orientation is not a suitable feature to use.

On contrast, density and distance insignificantly increased the accuracy rate only by 1%, which is technically a better result when compared to the previous combination, and despite only providing a slight increase in accuracy, this proves that the distance feature could be considered as an appropriate feature.

In addition, when orientation and distance features are combined together, there is a drastic incline to the accuracy when compared to the orientation alone, whereby the results give an increase of 11%. As mentioned before, from these results the distance feature seems to be a good supporting feature. Even though it failed to find convergence when classifying on its own, the distance feature does provide assistance in terms of increasing the accuracy rate when applied to other features.

When combining all of the features, the results show only a small margin of accuracy rate. Originally, the accuracy rate for the combination all of the features gave 64.29%, which is not significant, and technically, the orientation lowers the accuracy rating.

## 7. Future Improvements

In this section of my project, I will be discussing the improvements that could be applied into the system in order to further evaluated the system, and/or make the system robust by implementing new algorithms or even improving some of the algorithm used for the purposes of enhancing the performance.

### *7.1 Larger sample size*

Firstly, the notion of having enough sample size is a vague term and varies depending on the scenario, where there are situations that requires great amount of sample size in order to maximise performance. This is the case for my current system because of a larger sample size would cover multitude variations of the characters. These variations contribute into creating a classification that is more reliable when tested against new data.

However, larger sample size does not necessarily mean a higher accuracy rating. The purpose of having a larger sample size is to determine whether the accuracy rate is consistent. For example, the sample size could be divided into multiple portion of the same size and tested against new data to determine whether the accuracy have any drastic changes when compared with each sample size portions.

Theoretically, the accuracy rate should be consistent with each portion, but if this is not the case, then this means that there are some minor issues with either the classification or the features used or even a combination of both. These issues may include overfitting, where the system's classification scheme is too well trained to the training data, where the cost of error is low, which meant that when new data with different variations of the characters are introduced against it, the error rate may be significant.

### *7.2 Consecutive image processing for sample data*

With the time scale I had been given, there were algorithms which were paramount to the system, whereas this algorithm is not essential, but it would be useful to have. At the moment, I had been combining the sample data into one big image instead of reading the images individually, which would had been the ideal solution to the problem. As mentioned previously, the time-scale of the project is not long enough to have this feature implemented into the system, and would require additional processing such as systematically reading the images and appending the data, represented as records into a struct, whilst ensuring that the correct data is stored.

### *7.3 Adaptive thresholding*

One of the fundamentals of computer vision systems is choosing the correct threshold of a given problem. As for the project's case, a threshold needs to be selected in order to convert the image from a grayscale into a binary image. Images may have different thresholds, which depends on multiple conditions, such as the lighting in the environment, the camera used, or even the equipment used to write. In particular, when

comparing handwriting density with a pen and a pencil, the pencil would not have a pronounced appearance when compared to the pen.

The appropriate threshold can be manually obtained by 'trial and error', but the challenge is making it automated, which is not ideal when implemented into the real world context. My proposal approach to this problem is to use a histogram based on the grayscale version of the image. Given the histogram, we could analyse the graph to find the colour distribution of the pixels. The histogram has an x-axis which represents the colour of a pixel based on a scale, where 0 are the black pixels, 255 are the white pixels, whilst anything in between are gray pixels.

The proposed solution is only theoretical and had not been proven or tested, which means that the solution is not necessarily going to give the appropriate threshold. Additionally, based on this approach for this problem, the threshold is more of an estimate to what the threshold should be rather than what the exact threshold is for the image.

#### *7.4 Algorithm for joining fragmented characters*

As mentioned in the evaluation stage of the project, the algorithm which merges fragmented characters is not perfect, where there are problems such as if characters are closely written together, the algorithm would merge them and think of them as a single character. When this algorithm is applied to the real world context where handwritten characters are written closely to each other to represent a word, the algorithm would mistakenly read the whole word as one character.

Since this was an unforeseen problem, I had to quickly and efficiently come up with an algorithm to solve it, which meant that I needed a quick fix for the problem. Hence, there are probably better solutions compared to mine, where the solutions are efficient and robust.

However, my algorithm could be improved in order to make it more robust by detecting other aspects of fragmented characters. For example, if the area of the character's fragmented region is within a small range, then the algorithm could assume that it is in fact part of the character, assuming that there are no noise near the fragmented character. Selecting this range may be problem because the area of fragmented regions varies from character to character, such as the dot in an 'i' would have a small area, whilst fragmented characters presented from the testing section would have a large area. This means other factors need to be considered rather than the position and the area.

#### *7.5 Features: new features and performance*

Although the current features that I had used for the system are sufficient, there are better features which could be used. Currently, the features that I had used have problems with invariants. These variants include size and rotation. For example, when trying to classify a character that had been written at an angle and/or written at a smaller/larger size compared to the sample data, the system would not be able to

recognise what the character is. This shows that other features should be adopted, where these features are invariant against size and rotation.

Features that are invariant is when extracting the features from similar objects that had been rotated and of different sizes, should result in obtain the same feature points despite the structural difference between the objects. For example, when trying to detect a human face, there are more or less an infinite amount of variations, but these features are capable of detect the same feature points despite the variations, hence making a feature invariant. This would be the desired features as it would enable the system to become robust when tested against new data.

However, the current features implemented on the system are sufficient when invariants are not taken into consideration. In general, in the real world context as an assumption, rotation of a handwritten character is a rare case, where the individual won't necessarily write a character at an angle, or too much where the difference does not matter. Similarly, size is a more of a common case, but usually the sizes of the handwritten characters do not vary as much. As seen in my sample data, there are handwritten characters that differ in terms of sizes, but they do not differ as much.

## *7.6 Classification scheme*

The current classification scheme for my system is one of many. Originally, I had planned to experiment with different classifiers, but due to unforeseen problems, I had to allocate time to those instead, which meant that I was not able to try another classifier. The point of conducting experiments of trying out different classifier is because there is not a single classifier that is considered the 'best'. The performance of the classifier is highly dependent on the problem that you have, where applying a different classifier to my system may increase or decrease the accuracy rate.

At the moment, deep learning is considered as one of the trends in terms of machine learning. The idea of deep learning is to build a set of algorithms in machine learning that would be used to try and learn in multiple levels, where each level has a different abstraction in order to help putting some sense to data.

Deep learning usually uses neural networks, which is what I had planned on experimenting with, but was unable to do so due to the time constraints. Implementing neural network into the system would be another interesting approach towards the project, since it works completely different compared to SVM. It would be interesting to see the differences in output when using a neural network, whether if it increases the accuracy rating or not. With the correct adjustments towards the weights of the neural network, I think that this classifier would surpass the SVM approach.

## *7.7 Store the classification scheme*

At the moment, in order to classify data, the system must read the training data and build the classification scheme every single time you wish to classify data. To save processing time, the classification scheme should be stored instead of having to keep building it. I was unable to implement this feature due to the lack of time. However, the system does works without this implementation, but it would be useful to have, since

the processing time would be decreased by approximately half, especially if the training data is large. In other words, the system would only have to process the input image and then classify it. This was not a necessary requirement, but the purpose of it is to save processing time.

## *7.8 User Interface*

Initially, the system was planned to be implemented as a mobile application, which meant that it required a friendly user interface. Although I believe that the functionality of the system proved to be at most importance compared to the UI rather than the usability principles, which meant that I was not able to prioritise on developing a UI for the system because of the limited time.

To get a basic idea, the mobile application should consist of multiple functionalities such as:

- Allow to take a photo using the mobile phone's camera.
- Select an image from the mobile phone's gallery.
- Classify the image.
- Produce a text display of the results.
- Allow the text display to be saved as a text file.

When classifying the image, the system would use pre-made classification scheme, since the current system has to keep on building this scheme instead of creating and storing the classification scheme. In other words, the system would only need to process the user's image and classify it consecutively by using the classification scheme.

## 8. Conclusion

In conclusion, I believe that the achievements and success in this project are reflected based on the initial aims from the beginning. For this conclusion, I will reinstate the aims that started of the project as well as the specifications mentioned in the design stage of this report, and provide a depth discussion towards what the system had achieved and what could be learnt from it.

One of my initial aims was to investigate useful libraries in MATLAB. Currently, several components of the system comprise of important pre-built MATLAB functions. For example, regionprops is one of the most important components of the system, and the values, i.e. area, centroid, bounding box of the regions that it returned was used throughout the process of the system. I was able to fully comprehend the correct use of these functions and applied them towards the system based on the continuous investigation I had conducted throughout the process of implementation, in terms of learning about the functions.

Another aim of the project was to investigate current technologies that are relevant to my project. I believe that I had provided a comprehensible literature review on the articles that I had read. My knowledge of the problem evolved after reading about the problems and the solutions that had been proposed by current systems, where I had learnt a realistic perspective of the project. In other words, by reading the literatures, I was able to have achieved and learnt the background knowledge in order to enhance my approach to the solution, such as by clearly dictating what are the distinct stages of the system.

In terms of the aim referring to the future prospect of the system, I believe that the report covers the majority of the improvements that could be made if the system is to be adopted by others. Through the sheer amount of testing that had been done to the system, I learnt that the system is by no means perfect. The future prospects of the system are there in order to give an insight to what the system could become, given enough time to do the necessary implementation and testing.

However, at this stage, it is difficult to see all of the problems which may arise once these improvements had been implemented. This is applicable to numerous scenarios, but as a guideline, when implementing new sections into the system, one should automatically assume that there will be something bound to go wrong, which means that planning must be undertaken to mitigate or avoid these risks.

On a contrary, I believe that the end system's functionalities had met the requirements and acceptance criteria. For instance, the acceptance criteria for the pre-processing stage were to remove as much noise as possible, whilst attaining information of the characters. The unit testing stage proved that the system was able to achieve this requirement, whereby using a multitude of noise filtering techniques had managed to minimise the amount of noise present in the image.

When applying individual noise filtering techniques, each technique showed different results, where some techniques had remove that majority of the noise, but also removed information about the characters, and vice versa. The combination of the techniques

had almost nullify this trade-off as seen in the tests by removing the noise as well as some information of the characters, but further into the algorithm, the information is restored.

Moreover, one of the aim which states to distinguish individual characters had been achieved successfully. By using regionprops, I was able to obtain all of the regions, such as noise and characters that are present in the image. The system has an algorithm to filter out the regions which are considered as noise by using a range, whereby regions that have an area within the range are considered as characters.

The acceptance criteria for this aim were achieved where the algorithm extracts the information about the characters, i.e. area, centroid and location, but also remove the unwanted regions. Additionally, part of the criteria was to provide a visualisation of bounding boxes that contains the regions, which was shown in the unit tests that includes a visual representation of the minimum bounding box containing each valid characters that are present in the image as well as unwanted regions in a different bounding box colour.

The aim that refers to detecting features to uniquely identify each character had been accomplished by the system. Initially, I only had one feature which was the density feature, but since I had more time to implemented additional features, I had implemented the orientation and distance features. The performance of these features had been thoroughly evaluated, which resulted in the density feature providing the best results, whilst the other two features act more of a supporting role to give more reliable accuracy ratings, instead of drastically changing the accuracy rate.

From this, I had learnt that there are some features which provide different results, and that the features that I had been implemented into the system have some issues. For example, the features have problems when it comes to test data that is invariant, including rotation and scale. This meant that the system is not as robust, but when given the appropriate constraints then the system is more than capable of achieve good results.

One of the specifications was to label the sample data. The algorithm implemented for the end system was able to achieve this requirement, but this algorithm is specific to my sample data due to how it had been structured. The handwritten characters in the sample data are positioned uniformly. This meant that I knew which character a region is supposed to be based on their position.

The problem with this algorithm is that it is only applicable to my approach for gathering the sample data. However, this approach reinforces some constraints that is beneficial for me, where the form must be filled in a specific way. For example, the alphabets are to be written within the table, which meant that the sizes of the characters are of a limited range. In other words, the person filling the form should fit each of the characters in the corresponding position in the table.

According to my specification, the acceptance criteria for this function was to be able to 100% accurately label the regions, which had been achieved successfully, as seen in the



tests conducted. It was essential to get this algorithm to produce 100% accuracy otherwise the results would not be correct as the feature vectors would be mislabelled.

In terms of the classifier, I believe that the classifier implemented to the system is sufficient, and had met the requirements. On this note, the classifier's performance was appropriate for the problem, since the feature vector consisted of  $n$  dimensions, which is what SVM is capable of supporting, but was not able to support multiple class classification without adapting the code.

I think that this is the classifier's downfall, where the one against all relations models are not as accurate as they could be. From the tests, some of the results included no convergence, and the reason for this is because of how the models are created, and it does not help that there are 26 classes; meaning finding a convergence against 25 of those classes is difficult.

However, due to the time constraint, I was unable to experiment with other classifiers to compare if there are any margins in the results. Based on my literature reviews, I think that a classifier such as neural networks <sup>[13]</sup> would work effectively for this scenario. This is because of the properties of neural networks such as the weights, could be adjusted in order to produce better results.

## 9. Reflection

Over the course of this project, I was able to further develop my personal academia, and learnt important principles that could prove to be essential when undertaking future work. The concepts mentioned below are what I had considered as keys that enabled me to complete my project.

The ultimatum of the project is to find a solution to the problem. This problem solving element of the project enrich my knowledge in terms of developing my own approach, whilst using conventional means and applying my own originality. I had learnt that in order to solve a problem, the first step you must take is to understand the premises of the problem. By understanding the problem, I was able to build a realistic approach that was based on my capabilities and time constraints.

After conducting my literature reviews, I understood how to split the problem at hand into subtasks instead of approaching the problem as a whole. Each subtask gave a clear indication on what should be achieved, and at the end of it, these subtasks represented the system. Furthermore, the premises of the project is 'open-ended', and what I mean by this is that the problem could be approached in many ways, not just my solution. However, based on the research, I was able to narrow down these approaches, and by minimising the number of approaches I could perform an analysis to which approach is personally the most appropriate for me. Hence, I was able to adapt and re-design my specifications from the initial plan to make them realistic, and viable to my time scale.

One of the essential skills that stood out for me throughout the project is planning. The plan in the initial report did not take unforeseen problems under consideration, which meant that the scheduled inevitably failed. For example, I did not consider the task of having to developing an algorithm that would label the sample data, which required time that was supposed to be allocated to extracting features algorithm. Eventually, I was able to get back on track with my schedule once the problems were resolved, but in my future works, when developing a plan, I will take into consideration any possible problems that may arise and adapt my plan to them.

In terms of my academia, whenever faced with a project of this scale, I had usually worked with a group, and metaphorically, this project was presented as a blank canvas and I had the power to what goes into the canvas. From this project, although there were some guidance provided by my supervisor, this project was achieved on my own. As a result, I had learnt to be more independent to a certain degree, and since it is my project, I had the freedom to do what I want, but within reason. For example, the features that I had implemented into the system shows some originality. I decided to use these features instead of the conventional that had been mentioned in current systems because I had understanding on how to implement these features, as well as I wanted my project to be unique from the current systems.

Moreover, I had developed a sense of professionalism. For example, I had weekly meetings with my supervisor, where we discuss the progress of the project. In these meetings, I had usually prepared a strong agenda for my supervisor in order for him to get clearer idea of the progression, in regards to the project. Additionally, for every

meeting, I had made sure that I was on time, since I believe that punctuality is an important trait for building a strong and professional relationship with my supervisor.

Finally, I think that the most important principle that I had carried out throughout the process of this project is *consistency*. I was able to accomplish the achievements made from this project because I had allocated a time frame of 2-5 hours, per weekday. Usually, I have a set of minor goals that I should be able to complete from the given time frame. Through consistency, I was able to manage my time effectively, which allowed me to produce work efficiently, especially since I had other priorities other than the project. I would recommend to those who enrol on a project, whether small or large scale to work consistently because I believe that the end results would be better than the results when working inconsistently.

## 10. Appendix A

Following is a 7x26 table, where the outline of the table is barely visible, but enough to be seen by the human eye.

## 11. Appendix B

### 11.1 Density Vector

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0.353	0.462	0.437	0.697	0.613	0.412	0.44	0.134	0	0	0	0.277	0.546	0	0	0	0	0.168	0.429	0.462	0.61	0.445	0.471	0.37

### 11.2 Average Distance Vector

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
22.67	22.65	22.667	22.52	23.43	23.14	21.25	21.09	22.21	21.995	24.81	24.62	24.16	24.13	22.81	22.8	0	0	0	0	0	0	6.324	6.23

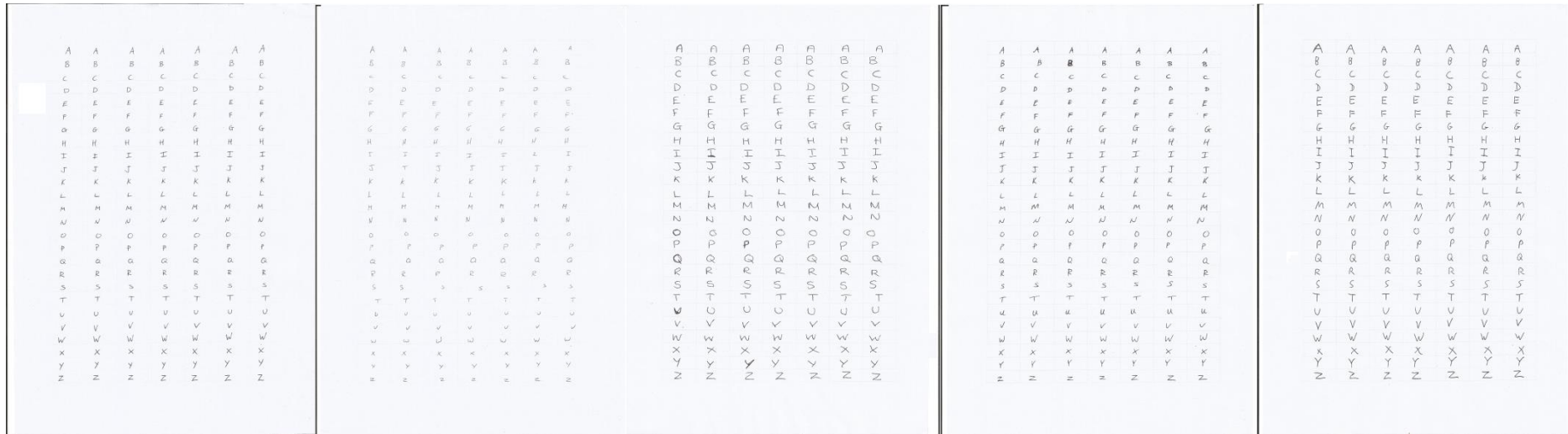
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
5.32	5.32	0	0	0	0	0	0	0	0	14.888	14.57	15.34	15.083	15.06	14.733	12.01	11.86	13.89	13.79	13.688	13.71	14.278	14.437

### 11.3 Orientation Vector

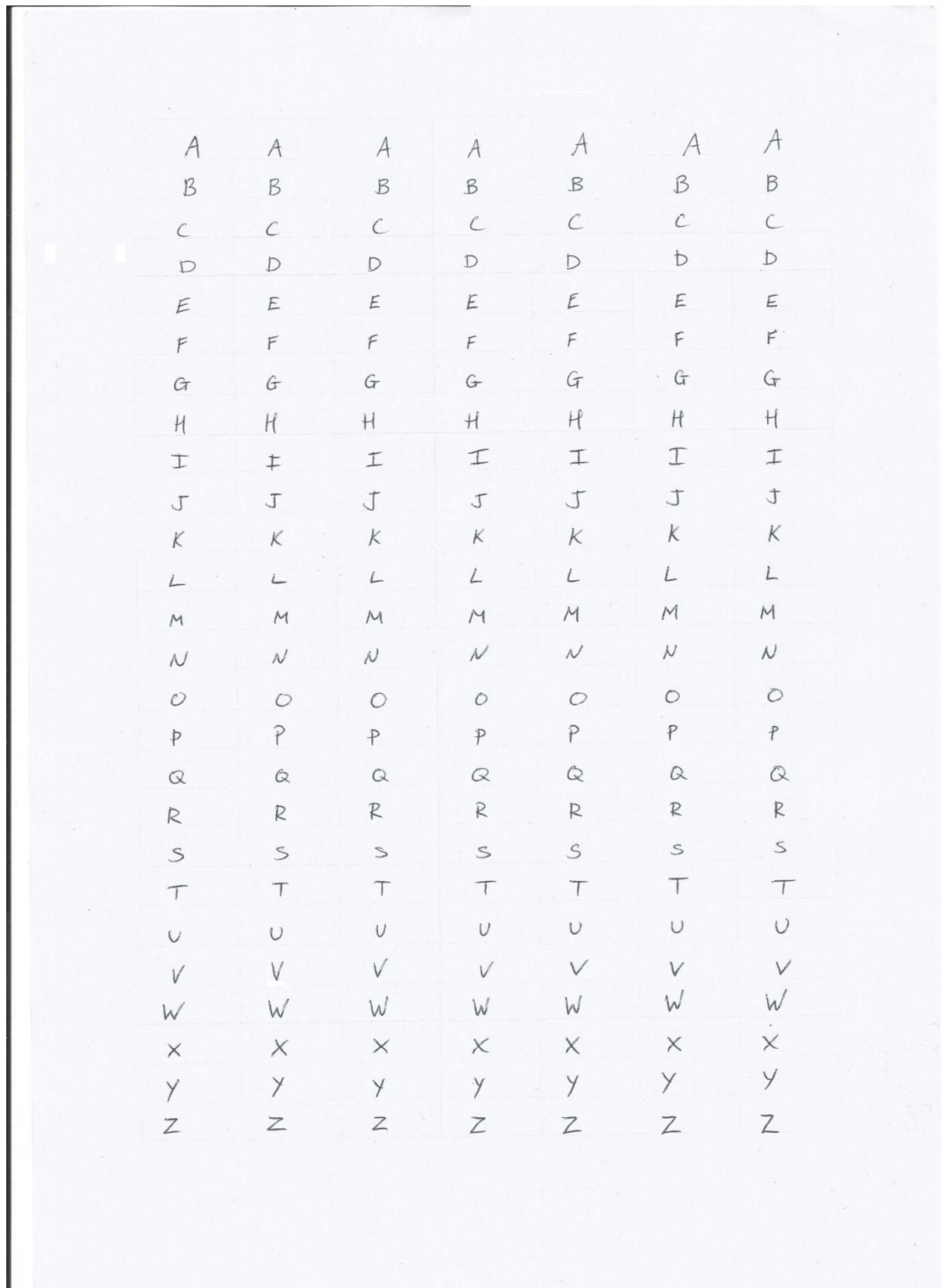
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
-17.1	26.14	14.02	-9.905	13.03	0	-33	-5.8	0	0	0	-2.46	-1.35	0	0	0	0	-7.44	-32.31	16.07	-4.48	22.03	0	-29.84

## 12. Appendix C

12.1 000.png



12.2 001.png



12.3 002.png

A	A	A	A	A	A	A
B	B	B	B	B	B	B
C	C	C	C	C	C	C
D	D	D	D	D	D	D
E	E	E	E	E	E	E
F	F	F	F	F	F	F
G	G	G	G	G	G	G
H	H	H	H	H	H	H
I	I	I	I	I	I	I
J	J	J	J	J	J	J
K	K	K	K	K	K	K
L	L	L	L	L	L	L
M	M	M	M	M	M	M
N	N	N	N	N	N	N
O	O	O	O	O	O	O
P	P	P	P	P	P	P
Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R
S	S	S	S	S	S	S
T	T	T	T	T	T	T
U	U	U	U	U	U	U
V	V	V	V	V	V	V
W	W	W	W	W	W	W
X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z



12.4 003.png

A	A	A	A	A	A	A
B	B	B	B	B	B	B
C	C	C	C	C	C	C
D	D	D	D	D	D	D
E	E	E	E	E	E	E
F	F	F	F	F	F	F
G	G	G	G	G	G	G
H	H	H	H	H	H	H
I	I	I	I	I	I	I
J	J	J	J	J	J	J
K	K	K	K	K	K	K
L	L	L	L	L	L	L
M	M	M	M	M	M	M
N	N	N	N	N	N	N
O	O	O	O	O	O	O
P	P	P	P	P	P	P
Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R
S	S	S	S	S	S	S
T	T	T	T	T	T	T
U	U	U	U	U	U	U
V	V	V	V	V	V	V
W	W	W	W	W	W	W
X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z

12.5 004.png

A	A	A	A	A	A	A
B	B	B	B	B	B	B
C	C	C	C	C	C	C
D	D	D	D	D	D	D
E	E	E	E	E	E	E
F	F	F	F	F	F	F
G	G	G	G	G	G	G
H	H	H	H	H	H	H
I	I	I	I	I	I	I
J	J	J	J	J	J	J
K	K	K	K	K	K	K
L	L	L	L	L	L	L
M	M	M	M	M	M	M
N	N	N	N	N	N	N
O	O	O	O	O	O	O
P	P	P	P	P	P	P
Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R
S	S	S	S	S	S	S
T	T	T	T	T	T	T
U	U	U	U	U	U	U
V	V	V	V	V	V	V
W	W	W	W	W	W	W
X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z

12.6 005.png

A	A	A	A	A	A	A
B	B	B	B	B	B	B
C	C	C	C	C	C	C
D	D	D	D	D	D	D
E	E	E	E	E	E	E
F	F	F	F	F	F	F
G	G	G	G	G	G	G
H	H	H	H	H	H	H
I	I	I	I	I	I	I
J	J	J	J	J	J	J
K	K	K	K	K	K	K
L	L	L	L	L	L	L
M	M	M	M	M	M	M
N	N	N	N	N	N	N
O	O	O	O	O	O	O
P	P	P	P	P	P	P
Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R
S	S	S	S	S	S	S
T	T	T	T	T	T	T
U	U	U	U	U	U	U
V	V	V	V	V	V	V
W	W	W	W	W	W	W
X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z



12.7 006.png

A	A	A	A	A	A	A
B	B	B	B	B	B	B
C	C	C	C	C	C	C
D	D	D	D	D	D	D
E	E	E	E	E	E	E
F	F	F	F	F	F	F
G	G	G	G	G	G	G
H	H	H	H	H	H	H
I	I	I	I	I	I	I
J	J	J	J	J	J	J
K	K	K	K	K	K	K
L	L	L	L	L	L	L
M	M	M	M	M	M	M
N	N	N	N	N	N	N
O	O	O	O	O	O	O
P	P	P	P	P	P	P
Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R
S	S	S	S	S	S	S
T	T	T	T	T	T	T
U	U	U	U	U	U	U
V	V	V	V	V	V	V
W	W	W	W	W	W	W
X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z

## 13. References

- [1] Thom Holwerda. 2013. History of handwriting recognition [Online]. Available at: [http://www.osnews.com/story/26838/Palm\\_I\\_m\\_ready\\_to\\_wallow\\_now/page2/](http://www.osnews.com/story/26838/Palm_I_m_ready_to_wallow_now/page2/) [Accessed: 26/04/2016]
- [2] Herbert F. Schantz. 1982. *The History of OCR: Optical Character Recognition*. 1<sup>st</sup> ed. University of Michigan: Recognition Technologies Users Association.
- [3] Chen, M.Y., Kundu, A. and Zhou, J., 1994. Off-line handwritten word recognition using a hidden Markov model type stochastic network. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(5), pp.481-496.
- [4] Kim, G. and Govindaraju, V., 1997. A lexicon driven approach to handwritten word recognition for real-time applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(4), pp.366-379.
- [5] Mori, S., Nishida, H. and Yamada, H., 1999. *Optical character recognition*. John Wiley & Sons, Inc..
- [6] Cleve Moler. 2004. The Origins of MATLAB [Online]. Available at: <http://uk.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html> [Accessed: 26/04/2016]
- [7] Poggio, T. and Cauwenberghs, G., 2001. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13, p.409.
- [8] Belongie, S., Malik, J. and Puzicha, J., 2002. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4), pp.509-522.
- [9] LaLomia, M., 1994, April. User acceptance of handwritten recognition accuracy. In *Conference companion on Human factors in computing systems* (pp. 107-108). ACM.
- [10] Marti, U.V. and Bunke, H., 1999, September. A full English sentence database for off-line handwriting recognition. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on* (pp. 705-708). IEEE.
- [11] Cao, J., Ahmadi, M. and Shridhar, M., 1995. Recognition of handwritten numerals with multiple feature and multistage classifier. *Pattern Recognition*, 28(2), pp.153-160.
- [12] Kimura, F. and Shridhar, M., 1991. Handwritten numerical recognition based on multiple algorithms. *Pattern recognition*, 24(10), pp.969-983.

- [13] Patil, V. and Shimpi, S., 2011. Handwritten English character recognition using neural network. *Elixir Comp. Sci. & Engg*, 41, pp.5587-5591.
- [14] Nasien, D., Haron, H. and Yuhaniz, S.S., 2010, March. Support vector machine (SVM) for english handwritten character recognition. In *2010 Second International Conference on Computer Engineering and Applications* (pp. 249-252). IEEE.
- [15] Narendra, P.M., 1981. A separable median filter for image noise smoothing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1), pp.20-29.
- [16] MathWorks, Inc. 2016. bwmorph [Online]. Available at: <http://uk.mathworks.com/help/images/ref/bwmorph.html> [Accessed: 20/04/2016]
- [17] MathWorks, Inc. 2016. rgb2gray [Online]. Available at: <http://uk.mathworks.com/help/matlab/ref/rgb2gray.html> [Accessed: 20/04/2016]
- [18] MathWorks, Inc. 2016. im2bw [Online]. Available at: <http://uk.mathworks.com/help/images/ref/im2bw.html> [Accessed: 20/04/2016]
- [19] MathWorks, Inc. 2016. bwconncomp [Online]. Available at: <http://uk.mathworks.com/help/images/ref/bwconncomp.html> [Accessed: 21/04/2016]
- [20] MathWorks, Inc. 2016. regionprops [Online]. Available at: <http://uk.mathworks.com/help/images/ref/regionprops.html> [Accessed: 21/04/2016]
- [21] Image Analyst. 2014. How to draw boundingBox/rectangle transparent over an image [Online]. Available at: <http://uk.mathworks.com/matlabcentral/answers/158151-how-to-draw-boundingbox-rectangle-transparent-over-an-image> [Accessed: 15/04/2016]
- [22] Rohitha Perera. 2011. All you Need to Know about Flowcharting [Online]. Available at: <http://creately.com/blog/diagrams/all-you-need-to-know-about-flowcharting/> [Accessed: 21/04/2016]
- [23] Purchase, H.C., Colpoys, L., McGill, M., Carrington, D. and Britton, C., 2001, December. UML class diagram syntax: an empirical study of comprehension. In *Proceedings of the 2001 Asia-Pacific symposium on Information visualisation- Volume 9* (pp. 113-120). Australian Computer Society, Inc.
- [24] MathWorks, Inc. 2016. medfilt2 [Online]. Available at: <http://uk.mathworks.com/help/images/ref/medfilt2.html> [Accessed: 04/04/2016]

- [25] Patidar, P., Gupta, M., Srivastava, S. and Nagawat, A.K., 2010. Image de-noising by various filters for different noise. *International Journal of Computer Applications (0975-8887) Volume*.
- [26] Otsu, N., 1975. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296), pp.23-27.
- [27] MathWorks, Inc. 2016. imhist [Online]. Available at: <http://uk.mathworks.com/help/images/ref/imhist.html> [Accessed: 14/03/2016]
- [28] Nomura, S., Yamanaka, K., Katai, O., Kawakami, H. and Shiose, T., 2005. A novel adaptive morphological approach for degraded character image segmentation. *Pattern Recognition*, 38(11), pp.1961-1975.
- [29] Cody Neuburger. 2012. Multi Class SVM [Online]. Available at: <http://uk.mathworks.com/matlabcentral/fileexchange/39352-multi-class-svm/content/multisvm.m> [Accessed: 04/02/2016]
- [30] MathWorks, Inc. 2016. imclearborder [Online]. Available at: <http://uk.mathworks.com/help/images/ref/imclearborder.html> [Accessed: 26/03/2016]