

https://dspace.org/

Extracting finite automata from hand-drawn images

Aruleba, Kehinde Daniel

2020
https://hdl.handle.net/10539/30989

Downloaded from a DSpace repository. https://dspace.org/

Extracting Finite Automata from Hand-drawn Images

Kehinde Daniel Aruleba



A thesis submitted to the
Faculty of Science,
University of the Witwatersrand, Johannesburg,
in fulfilment of the requirements for the Degree of
Doctor of Philosophy

Supervisors
Professor Sigrid Ewert
Professor Ian Sanders
Dr. Mpho Raborife

October 2020

Declaration

I declare that this thesis is my own, unaided work. It is being submitted for the Degree of Doctor of Philosophy at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

Signed: 

Date: October 2020

Abstract

The recognition of shapes in graphical and handwritten documents (such as architectural drawings, mathematical notations, and engineering drawings) is an intensive research activity in the field of pattern recognition and document analysis. The process of recognising hand-drawn diagrams is an easy task for humans, but a challenging task for the computer to process automatically due to many factors like the variation in individual handwriting, writing style, and pen ball tip. In this work, we focus on how to use an offline recognition approach to describe, classify, recognise and generate transition tables for finite automaton (FA) diagrams drawn by students. To do this, we have investigated the different areas of handwriting recognition, with the aim of gaining a better understanding of this research area. Based on our findings, there is no existing dataset or system that uses offline recognition for recognising FA diagrams. This has led to the creation of our datasets (i.e. DB_a and DB_b) of hand-drawn FA diagrams. These datasets were used for the entire recognition process. In this thesis, we have introduced a multi-phase recognition algorithm. At each phase, the system takes as input the output from the previous phase. Different pre-processing techniques were performed on an input FA diagram. We showed how bounding box segmentation using the structure of an FA diagram and the relationship that exists between the components of an FA diagram can be done. Also, this thesis explains why the bounding box segmentation approach will give a better result compared to segmentation at the stroke level. We went further to classify the components of an FA diagram to their unique classes. Evaluation of the recognition process was done at the structure level (i.e. stroke and symbol levels). At the stroke level, the algorithm achieved an accuracy of 95.4% and 94.0% at the symbol level on DB_a while it achieved an accuracy of 89.8% and 87.6% at both levels respectively on DB_b. Also, the system transition table generator achieved an accuracy of 86.1% and 84.4% on DB_a and DB_b respectively. The idea presented in this study can aid academics teaching Automata Theory in assessing students' exercises and examinations if implemented as a software prototype.

Acknowledgement

I owe my deepest gratitude to my supervisors, Professor Sigrid Ewert, Professor Ian Sanders and Dr Mpho Raborife for their advice, mentorship and for giving me the opportunity to work under their guidance throughout the past three years. Their rich knowledge and experience always make me feel that there could not have been a better supervisor. Their scrutiny of my work has improved my critical analytical thinking for research and academic writing skills.

I am indebted to my sponsors CSIR DST-Interbursary Support (IBS) and Wits post-graduate and financial aid office for offering me a post-graduate merit award. I would like to appreciate the students who volunteered to draw the finite automaton diagrams used in this thesis and also Professor Sigrid and Kgomotso for giving me access to formal language and automata theory past examination papers. I would also like to show gratitude to my friends and colleagues, Folakemi, George, Blessing, Mustapha, Michael, Kelvin, Alex, Nuru, Olaperi, Ronny, Jeremiah, Adulsalam, Rediet, Sara and my sincere apologies for anyone I have forgotten to mention for their encouragement and motivation. I would also like to acknowledge the help of Marike Kluyts in writing this thesis.

Last but certainly not least, thanks go to my family (Mr and Mrs Aruleba, Dr Oluwaseyi Aruleba, my twin - Taiwo Aruleba, Idowu Aruleba and Oluwadabira Adebayo) for they made this fulfilling experience possible through their constant support. They have been of great support when I encountered difficulties, from the application stage to the end of the thesis.

Contents

Declaration i

Abstract ii

Acknowledgement iii

Contents iv

List of Figures vii

List of Tables x

Publications xi

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Motivation and Problem Statement . . . 4
- 1.3 Research Aim and Objectives . . . 4
  - 1.3.1 Research Aim . . . 4
  - 1.3.2 Research Objectives . . . 4
- 1.4 Key Contribution of the Research Study . . . 5
- 1.5 Scope and Limitation . . . 6
- 1.6 Thesis Organisation . . . 6

2 Background and Related Work 8
- 2.1 Introduction . . . 8
- 2.2 Pattern Recognition . . . 9
  - 2.2.1 Supervised Learning . . . 12
    - 2.2.1.1 Classification . . . 13
    - 2.2.1.2 Regression . . . 14
  - 2.2.2 Unsupervised Learning . . . 14
- 2.3 Sketch Recognition . . . 16
  - 2.3.1 Sketch Recognition Techniques . . . 17
    - 2.3.1.1 Template Matching . . . 19
    - 2.3.1.2 Statistical Classification . . . 21
    - 2.3.1.3 Syntactic Matching . . . 22
    - 2.3.1.4 Neural Networks . . . 23
- 2.4 Image Processing Techniques used in Sketch Recognition . . . 23

2.4.1 Data Acquisition . . . 24
2.4.2 Noise Removal . . . 24
2.4.2.1 Gaussian Noise . . . 25
2.4.2.2 Salt-and-pepper Noise . . . 26
2.4.3 Filtering . . . 26
2.4.3.1 Mean Filter . . . 27
2.4.3.2 Median filter . . . 27
2.4.4 Thresholding . . . 28
2.4.5 Image Segmentation . . . 33
2.4.6 Thinning . . . 35
2.4.6.1 Zhang-Suen Parallel Thinning Algorithm . . . 37
2.4.6.2 Thinning with Structuring Elements . . . 39
2.4.7 Chaincode . . . 42
2.5 Application Domain of Sketch Recognition . . . 44
2.5.1 Unified Modelling Language . . . 45
2.5.2 Flowchart . . . 46
2.5.3 Graph Diagram . . . 47
2.5.4 Other Recognition Systems . . . 48
2.6 Handwriting Recognition . . . 53
2.6.1 Types of Handwriting Recognition . . . 54
2.6.1.1 Online Recognition . . . 54
2.6.1.2 Offline Recognition . . . 55
2.7 Handwriting Databases . . . 56
2.7.1 IAM Database . . . 57
2.7.1.1 IAM Handwriting Database . . . 57
2.7.1.2 IAM On-Line Handwriting Database . . . 57
2.7.2 MNIST Database . . . 57
2.7.3 Flowcharts . . . 57
2.8 Techniques used in Handwriting Recognition . . . 58
2.8.1 Support Vector Machines . . . 58
Optimal Hyperplane Algorithm . . . 59
2.8.2 Recurrent Neural Networks . . . 60
2.8.2.1 Bidirectional Long Short-Term Memory . . . 61
2.9 Summary . . . 63

3 Methodology **65**
3.1 Introduction . . . 65
3.2 Data Acquisition . . . 66
3.2.1 Synthetic Data . . . 67
3.2.2 Raw Data . . . 68
3.3 Pre-processing . . . 69
3.4 Text and Graphics Separation . . . 70
3.5 Stroke Processing . . . 71
3.6 Segmentation . . . 71
3.7 Arrow Recognition . . . 72
3.8 Shape Detection and Recognition . . . 73
3.9 Text Detection and Recognition . . . 73

3.10 Performance Evaluation . . . 73
3.11 Conclusion . . . 74

4 Pre-processing of Hand-drawn FA Diagrams 76
4.1 Introduction . . . 76
4.2 Image Enhancement . . . 78
4.3 Noise Removal . . . 80
4.4 Thresholding . . . 81
4.5 Thinning . . . 83
4.6 Summary . . . 86

5 From Stroke to Finite Automaton 87
5.1 Introduction . . . 87
5.2 Text and Graphics Separation . . . 87
5.3 Stroke Processing . . . 92
5.4 Segmentation . . . 99
5.5 Arrow Detection and Recognition . . . 107
5.6 Shape Detection and Recognition . . . 111
5.7 Text Detection and Recognition . . . 113
5.8 Final Recognition Result . . . 114
5.9 FA Transition Table . . . 114
5.10 Summary . . . 117

6 Experimental Results and Discussion 119
6.1 Introduction . . . 119
6.2 Datasets . . . 120
6.3 Database . . . 120
6.4 Experiment . . . 121
6.5 System Analysis . . . 122
6.6 Misclassification . . . 126
6.7 Findings . . . 128

7 Conclusion and Future Work 130
7.1 Conclusion . . . 130
7.2 Future Work . . . 132

A Sample Data 154
A.1 Synthetic Data . . . 154
A.2 Raw Data . . . 156

B Participants Consent Letter 158
B.1 Student Letter . . . 158
B.2 Ethics Certificate . . . 159

C Data Annotation Tool 160

List of Figures

1.1 FA diagrams with different handwritings and drawing styles 1
1.2 Data acquisition modes 3
1.3 Components of an FA diagram 3

2.1 Different types of picture patterns 10
2.2 Components of a pattern recognition system [Essays, 2018] 11
2.3 Type of learning 12
2.4 Binary classification and multi-class classification [Bisht, 2019] 13
2.5 Examples of images that have been used in classification problems . . . 15
2.6 Linear regression [Gandhi, 2016] 16
2.7 Chemical structure interpretation process [Ouyang and Davis, 2011] . . 17
2.8 Graphical representations of objects in some applications 18
2.9 (a) The templates are marked in green over the input images (b) The target images are marked with the detection result [Dekel et al., 2015] . . 20
2.10 Statistical pattern recognition model [Jain et al., 2000] 21
2.11 Syntactic pattern recognition model [Albus et al., 2012] 22
2.12 Neural network architecture [Patterson and Gibson, 2017] 23
2.13 Images affected with salt-and-pepper and their corresponding output after filtering [Fu et al., 2019] 26
2.14 Image pixels affected by noise 27
2.15 First 3×3 window of Figure 2.14 27
2.16 Pixels of Figure 2.14 after applying median filter on its first 3×3 window 28
2.17 Thresholding [Mallick, 2015] 29
2.18 Thresholding a 5×4 image 32
2.19 Normalised histogram of Figure 2.18(a) 32
2.20 Text and Graphics Separation 33
2.21 Text and Graphics Separation 36
2.22 Thinning of a binary diagram [Robert et al., 2003] 36
2.23 Nine pixels in a 3×3 window 37
2.24 Counting the 01 patterns in the ordered sequence $P_2, P_3, ..., P_9$ [Zhang and Suen, 1984] 38
2.25 Hovering over an image with structuring elements 39
2.26 Sequence of rotated structuring elements used for thinning 40
2.27 Applying B_1 on input image A 41
2.28 Output of applying element B_1 on input image A 41
2.29 Applying B_2 on input image A_1 42
2.30 Applying B_5 on input images A_3 and A_4 43
2.31 Applying B_6 on input image A_4 43

2.32 Hand-drawn UML diagram recogniser . . . 46
2.33 Graphical user interface of OGR [Auer et al., 2013] . . . 48
2.34 Sketches in InkKit Interface [Chung et al., 2005] . . . 51
2.35 Handwriting processing categories . . . 53
2.36 A simple linear Support Vector Machine [Tong and Koller, 2001] . . . 59
2.37 RNN versus FNN [De Mulder et al., 2015] . . . 61
2.38 Recurrent Neural Network [Zhang et al., 2018] . . . 61
2.39 General structure of the bidirectional recurrent neural network (BRNN) shown unfolded in time for three time steps [Schuster and Paliwal, 1997] 62

3.1 Overview of the recognition system . . . 67
3.2 A sample FA input RGB diagram from DB_a . . . 69
3.3 Result of performing pre-processing on Figure 3.2 . . . 70
3.4 Separation of the text and graphical components of Figure 3.3 . . . 70
3.5 Detecting and recognising the text in Figure 3.3 . . . 74

4.1 Input RGB FA diagram . . . 77
4.2 Input FA diagram after conversion to grayscale . . . 78
4.3 Input image C . . . 78
4.4 Final output after histogram equalisation . . . 80
4.5 Figure 4.2 affected with two types of noise . . . 81
4.6 Figure 4.5(a) after applying median filter . . . 81
4.7 Thresholding of Figure 4.2 . . . 83
4.8 Hand-drawn FAs with different emphasis . . . 83
4.9 Zhang-Suen Algorithm on Figure 4.8 . . . 84
4.10 Output of the thinning algorithm . . . 86

5.1 Sample handwritten document from the IAM handwriting database . . . 90
5.2 Stroke classification accuracy for the DB_a images in relation to bias . . . 91
5.3 Stroke classification accuracy for the DB_b images in relation to bias . . . 91
5.4 Thinned FA diagrams and their corresponding reconstructed strokes . . . 92
5.5 Scan direction . . . 93
5.6 The transition when just one pixel is available . . . 94
5.7 Example of extraction when only one pixel is available . . . 96
5.8 Pixel P with more than one neighbour . . . 97
5.9 Circle extraction . . . 99
5.10 Circle extraction process . . . 99
5.11 Stroke level segmentation approach . . . 101
5.12 Bounding box segmentation approach . . . 101
5.13 FA diagram to be segmented . . . 102
5.14 The result of experiments performed over DB_a and DB_b to get an optimal value of β coefficient. . . . 104
5.15 Post-segmentation . . . 104
5.16 Noisy FA diagrams . . . 106
5.17 Performance comparison of our datasets. . . . 107
5.18 Types of arrows in an FA diagram . . . 108
5.19 Drawing of an arrow . . . 108
5.20 Example of varieties of an arrow shaft . . . 109

5.21 Features used for arrow recognition . . . 109
5.22 Single-stroke arrow . . . 110
5.23 Divided arrow stroke i.e. sub-strokes . . . 110
5.24 Sub-strokes classes . . . 111
5.25 Text detection and recognition in an FA diagram . . . 113
5.26 Output of all recognised components of the input FA diagrams . . . 115
5.27 An FA diagram . . . 116

6.1 Sample FA diagrams from DB_{on} . . . 125
6.2 Illustrations of misclassification from our datasets . . . 127
6.3 Arrow retraced . . . 127

A.1 Sample binarised input images . . . 154
A.2 Recognised FA diagrams . . . 155
A.3 LaTeX generated version of the first FA . . . 156
A.4 LaTeX generated version of the second FA . . . 156
A.5 Sample binarised input images . . . 157

C.1 Screenshot of the application for data annotation, including the annotation tools. . . . 160
C.2 Screenshot of the application for data annotation showing the annotation of some individual entities. . . . 161

List of Tables

2.1 Pattern recognition models [Jain et al., 2000] . . . 19

3.1 FA datasets summary . . . 68

4.1 Histogram equalisation process of Figure 4.3 . . . 80

5.1 Transition table representation of Figure 5.27 . . . 116

6.1 FA datasets . . . 122
6.2 FA database overview . . . 122
6.3 Running time in milliseconds taken by each phase. . . . 123
6.4 Recall of stroke labeling accuracy and symbol recognition . . . 123
6.5 Recognition system accuracy at stroke and symbol level . . . 124
6.6 Comparison of our datasets and DB_{on} . . . 125
6.7 Total running time in seconds . . . 126
6.8 Accuracy of each symbol class label . . . 126
6.9 Analysis presenting the effect of each phase on the entire recognition process on our databases . . . 129

Publications

Some ideas in this thesis – including verbatim images, equations, and concept descriptions have featured in the following peer-reviewed publications (conference and workshop).

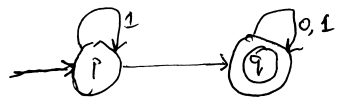
- Kehinde Aruleba, (2019) "Offline Approach for Recognising Hand-drawn Finite Automata Images". Presented at the third Black in AI workshop co-located with Conference on Neural Information Processing Systems, Vancouver Convention Center, Vancouver, Canada.
- Kehinde Aruleba, Sigrid Ewert, Ian Sanders and Mpho Raborife, (2018) "Preprocessing and Feature Extraction Techniques for Hand-drawn Finite Automata". In the Proceedings of IST Africa (IEEE), Gaborone, Botswana.
- Kehinde Aruleba, (2018) "From Stroke to Finite Automata". Presented at the second Black in AI workshop co-located with Conference on Neural Information Processing Systems, Palais des Congrès de Montréal, Montréal, Canada.

Chapter 1

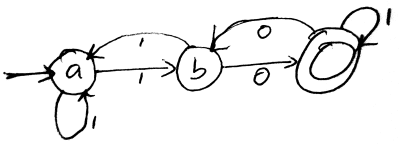
Introduction

1.1 Introduction

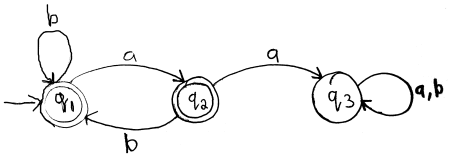
Sketching with pen and paper is generally a major part of design-related fields such as engineering, computer science, and architecture. This form of casual communication is quick and easy and allows developers to focus on the main structural issues rather than irrelevant details. This has led to the development of sketch recognisers for different fields that deal with designs. Designing of a sketch recogniser for new domains is a serious and time-consuming task. This is because free-hand sketching lacks accuracy and varies from person to person, as shown in Figure 1.1.



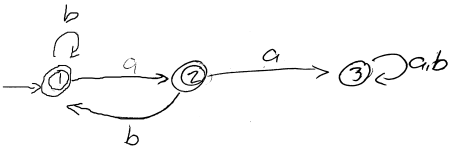
(a) Example of FA drawn by student A



(b) Example of FA drawn by student B



(c) Example of FA drawn by student C



(d) Example of FA drawn by student D

Figure 1.1: FA diagrams with different handwritings and drawing styles

The field of handwritten pattern recognition has been in existence for over seventy years [Graves and Schmidhuber, 2009; Jiang et al., 2016]. Research in this field started from isolated text characters and numbers or digits. The recognition focus then shifted to words. Recognition of cursive words was and still is considered a cumbersome task when compared to character recognition, due to the following reasons;

- There are many words in a language compared to characters.
- The process of segmenting a handwritten word image into text characters is challenging due to the cursive nature of text.

The recognition process of handwritten documents then advanced to sentence recognition and progressively to the development of end-to-end recognition systems that take input such as texts, words and images, and process it to produce an improved version of it as an output. These have led to the wide adoption of electronic forms of communication in the last decades.

Due to the wide acceptance of this form of communication, digitisation of information has spread into numerous domains, making research in pattern recognition shift gradually from handwritten document analysis and recognition (plain text) to more structured sketch input recognition. Sketch recognition and interpretation are challenging and may be unrealistic with differences in images and handwritings [Mahoney and Fromherz, 2002]. However, focusing on designing a recognition system for a specific domain is more realistic than designing a single system for multiple domains [Alvarado, 2007]. This approach led to the development of some recognition frameworks which are domain-specific for example, mechanical engineering [Alvarado and Davis, 2006], Unified Modelling Language (UML) class diagrams [Hammond and Davis, 2006; Lank et al., 2000], Webpage design [Lin et al., 2000] and Graphical User Interface (GUI) design [Landay and Myers, 1995, 2001; Hammond and Davis, 2005].

The process of recognising handwritten documents can be classified into two, namely; online and offline recognition (see Figure 1.2). Numerous online recognition systems running on pen input personal computers (PCs) have been proposed recently. These systems allow the creation of diagrams by drawing on the PC tablet using a stylus, i.e. it is real-time diagram recognition, where every stroke is captured as it is drawn. On the other hand, research in offline recognition approaches is still lagging due to the processes this approach is based on, i.e. after the image has been completely drawn by hand on paper, the system takes the diagram as a whole for processing.

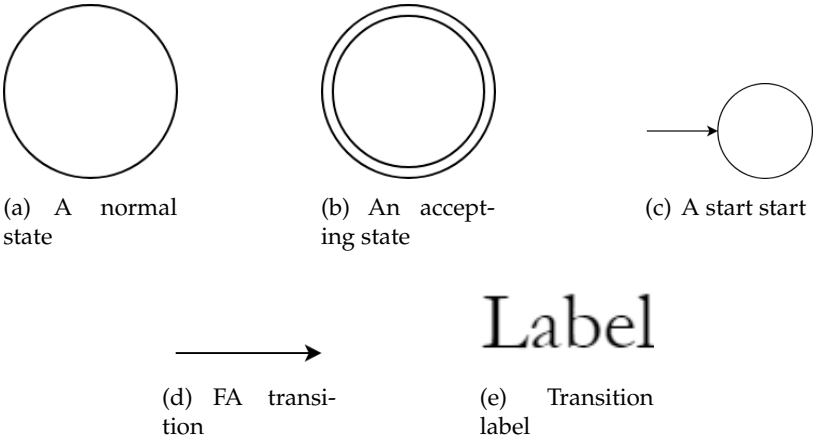


(a) Online data acquisition and recognition

(b) Offline data acquisition and recognition

Figure 1.2: Data acquisition modes

In this research, we focus on the pre-processing, extraction, classification and recognition of hand-drawn Finite Automaton (FA) diagrams using an offline recognition approach. FA diagrams generally have their semantic features (components) in a simple visual form consisting of lines, arrows, circles and text as illustrated in Figure 1.3. States are represented as circles and transitions as arrows that connect different states to each other. One state of the FA is designated as the start state. This state is mostly drawn as a single circle with an arrow pointing to it. Moreover, a subset of states in an FA can be defined as accepting states and are usually drawn as two concentric circles (see Figure 1.3(b)).



(a) A normal state

(b) An accepting state

(c) A start start

(d) FA transition

(e) Transition label

Figure 1.3: Components of an FA diagram

1.2 Motivation and Problem Statement

Most users in diagrammatic domains use structured graphics editors such as Microsoft Visio [Halpin et al., 2003]. These editors are very efficient and powerful, but they can be difficult to use due to the fact that the space required for drawing is reduced by the menu area and the symbol icon area. Furthermore, moving a mouse around might be tiring and time-consuming. Research conducted by Refaat et al. [2008] compared the time that users require to draw a diagram with editors such as Visio and LaTeX to that of a recognition system developed for hand-drawn structured diagram recognition (i.e. offline recognition system). The results of their research study show that hand-drawn structured diagram recognition systems are time-saving with an average of 90% less processing and drawing time compared to graphics editors. This result indicates that designing an electronic system for hand-drawn recognition processes will save users time and hence be more beneficial compared to structured graphics editors.

The purpose of this research is to produce a sketch recognition technique that extracts an FA diagram, understands the extracted FA and translates it to its transition table. This will contribute to resolving some of the problems faced by FA instructional staff. According to Sanders (personal communication, October 2016), a problem faced by FA instructional staff in online learning courses is that some students take pictures of their hand-drawn FAs with mobile cameras and send them to the instructional staff by email for assessment. This becomes cumbersome for the staff when receiving thousands of emails from students. This makes the grading of the assessments difficult and time-consuming. To address this problem, this study has proposed a new approach to process these FA diagrams. To make the new approach really useful, we had to solve the problem of variability in handwriting and FA drawing styles of different students.

1.3 Research Aim and Objectives

1.3.1 Research Aim

The aim of this research work is to present an offline recognition approach to recognise hand-drawn FA diagrams and generate their corresponding transition tables.

1.3.2 Research Objectives

The specific objectives of this research are to;

(i) Extensively study and investigate the field of pattern recognition with a main focus on dataset acquisition, handwriting recognition, offline handwriting recognition and document recognition systems.

(ii) Develop a new approach using pattern recognition techniques to process, extract, classify and recognise hand-drawn FA diagrams.

(iii) Evaluate the developed recognition system.

1.4 Key Contribution of the Research Study

According to our knowledge, this study is the first attempt at recognising hand-drawn FA diagrams using offline recognition approaches and generating the corresponding transition tables. Below are the contributions of this thesis:

(i) Due to the lack of annotated data in the field of offline handwritten diagram recognition, we have created and annotated two offline datasets for FA diagrams. We showed that using these diagrams, the developed recognition system can achieve state-of-the-art results.

(ii) Due to the domain of this study, we adapted the thinning algorithm proposed in Boudaoud et al. [2015] for FA diagrams.

(iii) A new segmentation technique based on the bounding box approach was introduced. Using the proximity or closeness of a symbol that is wrongly segmented (i.e. a single symbol divided into two symbols during segmentation), we introduced a post-segmentation algorithm that joins these two symbols to form the original symbol.

(iv) The arrow detector and recogniser in Kara and Stahovich [2007] was extended in this study. The reason for extending the recogniser in Kara and Stahovich [2007] is to allow different shapes of arrow shafts and arrow drawn as two consecutive strokes to be recognised.

(v) We also generated the transition table of an input FA diagram, after the diagram has passed through all the recognition stages.

Most of the recognition approaches used in this thesis are strictly for FA diagram components, structures and the inter-relationships that exist between an FA diagram's components. Finally, we reported the accuracy and effectiveness of the recognition system with respect to the ground-truth data.

1.5 Scope and Limitation

The key limitation of this work and in handwriting recognition generally is the variation in handwriting from person to person, and different writing and drawing styles. Human handwriting varies, and different conditions (i.e. walking, being sad, and writing in a rush) can determine the way a person writes at a particular moment. Hence, this research cannot be absolute in that area. However, since this research focuses mainly on FA diagrams and the data used during the experiment was all collected from students, the collected data was sufficient for the purposes of this work.

Another issue is segmentation during offline recognition. Segmentation of images acquired offline has been regarded to be a cumbersome and challenging task [Alvaro et al., 2014; Zhang et al., 2017]. If the image is wrongly segmented, the accuracy of the classification algorithm used will be low. The segmentation approach described in this thesis focuses and works on FA structures and not necessarily other connected diagram types such as flowcharts and UML diagrams. The purpose of focusing the segmentation approach on FA diagrams only is to make the segmentation algorithm precise and domain-specific.

The last issue was during the evaluation and experiment stage. As said earlier, research in the field of offline recognition is still limited compared to online recognition. Hence, there are limited datasets in offline recognition. During the evaluation phase, there was no existing offline recognition system for FA diagrams to compare this work with.

1.6 Thesis Organisation

This thesis is divided into chapters. Chapter 1 presents an introduction of the thesis. Also, the research aim and the objectives required to achieve the aim were stated in this chapter. Since the focus of this thesis is on offline recognition of hand-drawn FA images, the background and related work presented in Chapter 2 focus mainly on the approaches and techniques used in this thesis. The chapter started by introducing a general idea in pattern recognition and then went further to investigate sketch recognition and its techniques. Also, some existing recognition systems, techniques and databases were discussed in this chapter and why they will fail to solve our research problem is also explained. Chapter 3 presents our methodology and data sets. Chapter 4 presents the pre-processing of the data. Since these data are gathered using an offline approach, they become noisy during acquisition. Hence, we introduced this chapter. The stroke

recognition process was addressed in Chapter 5. A new stroke extraction algorithm and a segmentation approach were discussed in this chapter. Segmentation of graphic components from textual components was introduced in this chapter. Classification of the different components of an FA diagram was also done here. It is always important to evaluate a recognition system. The evaluation of the system was presented in Chapter 6. We compared the recognised diagrams with the ground truth. This was done at stroke and symbol level. Also, system analysis and misclassification were reported in this chapter. Chapter 7 concludes the thesis, analysing how the recognition techniques can improve and contribute to the area of handwriting recognition, and gives ideas for further studies and improvements.

Chapter 2

Background and Related Work

2.1 Introduction

As discussed in Chapter 1, the aim of this study is to present an offline recognition approach to recognise hand-drawn FA diagrams and generate the recognised diagram's transition tables. The objectives to achieve this aim were also explained in the chapter. In this chapter, we will discuss pattern recognition, hand-drawn shape recognition techniques and algorithms that have relevance to our research. Furthermore, we also discuss some related works. The works that we consider herein have used a variety of recognition and extraction approaches with regard to hand-drawn diagrams.

Recognition of diagram notations by computer is an active area of research in Computer Science. Diagram recognition systems can be distinguished based on whether the system takes online input [Ozawa et al., 2009] in an interactive manner from the system user, or performs image processing on scanned paper documents obtained offline [Graves and Schmidhuber, 2009]. Research into creating practical computer recognition systems for the online and offline recognition of diagram notations such as mathematics notation [Smithies et al., 1999], music notation [Stuckelberg and Doermann, 1999], engineering drawings [Hammond and Davis, 2005], flowcharts [Bresler et al., 2014], finite automata [Babalola, 2015] and UML diagrams [Hammond and Davis, 2006] is ongoing.

During online recognition, the user gets immediate real-time feedback while drawing and recognising the system. Also, the user has the privilege of adjusting their writing and drawing style to ensure accurate recognition. Important features such as timestamps, tilts, and pressure can be extracted easily while using this type of system. All this makes the result obtained during online recognition more accurate when compared

to an offline system because it is assisted by an electronic device like tablet PCs. The field of online recognition of hand-drawn connected diagram has gain lots of attention in recent times. Most of the work done in this field focus on flowchart diagrams [Awal et al., 2011; Miyao and Maruyama, 2012; Chen et al., 2015b] and only a few on other types of connected diagrams [Tang et al., 2014; Karasneh and Chaudron, 2013]. These existing works use datasets that are domain-specific during training. Also, each connected diagram domain has a unique component(s). Due to these reasons, we cannot use the systems proposed in these works for our research.

The present research focuses on offline recognition of FAs since the recognition is performed on scanned or photographed paper documents, i.e. hard copy. A major drawback in this type of recognition compared to online recognition is the lack of temporal data in diagrams obtained using offline approaches. These data are not available because the computer only sees the drawn diagrams and not the process and steps the user took while drawing the diagram. Processes such as location, ordering, pen movement, and pressure of each of the points laid by the pen provide a number of features that can be used during computation. These processes and features are noted, computed, and saved during the actual drawing of the diagram in an online recognition system which makes this type of recognition easier. However, the process of knowing the location, ordering or pen movement is cumbersome in offline as these processes are not visible to the computer. Also, offline recognition systems can be computationally intensive and time-consuming for a user that will have to write or draw on a paper first before taking a photograph of the writing and running it on the system. Other drawbacks like the noise picked up while taking a photograph of the diagram itself can be misinterpreted as a diagram data, hence leading to difficulties for the entire recognition system. This can also lead to false positives during recognition. All these drawbacks make research in offline handwriting recognition still challenging and lagging behind when compared to online handwriting recognition. In this research, we will address all of these when developing the FA recognition system. Section 2.6.1 discusses more related works on online and offline recognition systems.

2.2 Pattern Recognition

The automatic description, clustering, classification and recognition of patterns are important research problems in many science and engineering fields such as medicine, computer vision and chemistry [Jain et al., 2000]. A pattern can be an entity of focus or interest which needs to be identified or recognised. Patterns can be any of these: a typewritten or handwritten character, a gesture, sound wave, fingerprint, a 2D or 3D

shape, tree, face, bar code or character image, and a human face. Figure 2.1 shows some patterns.

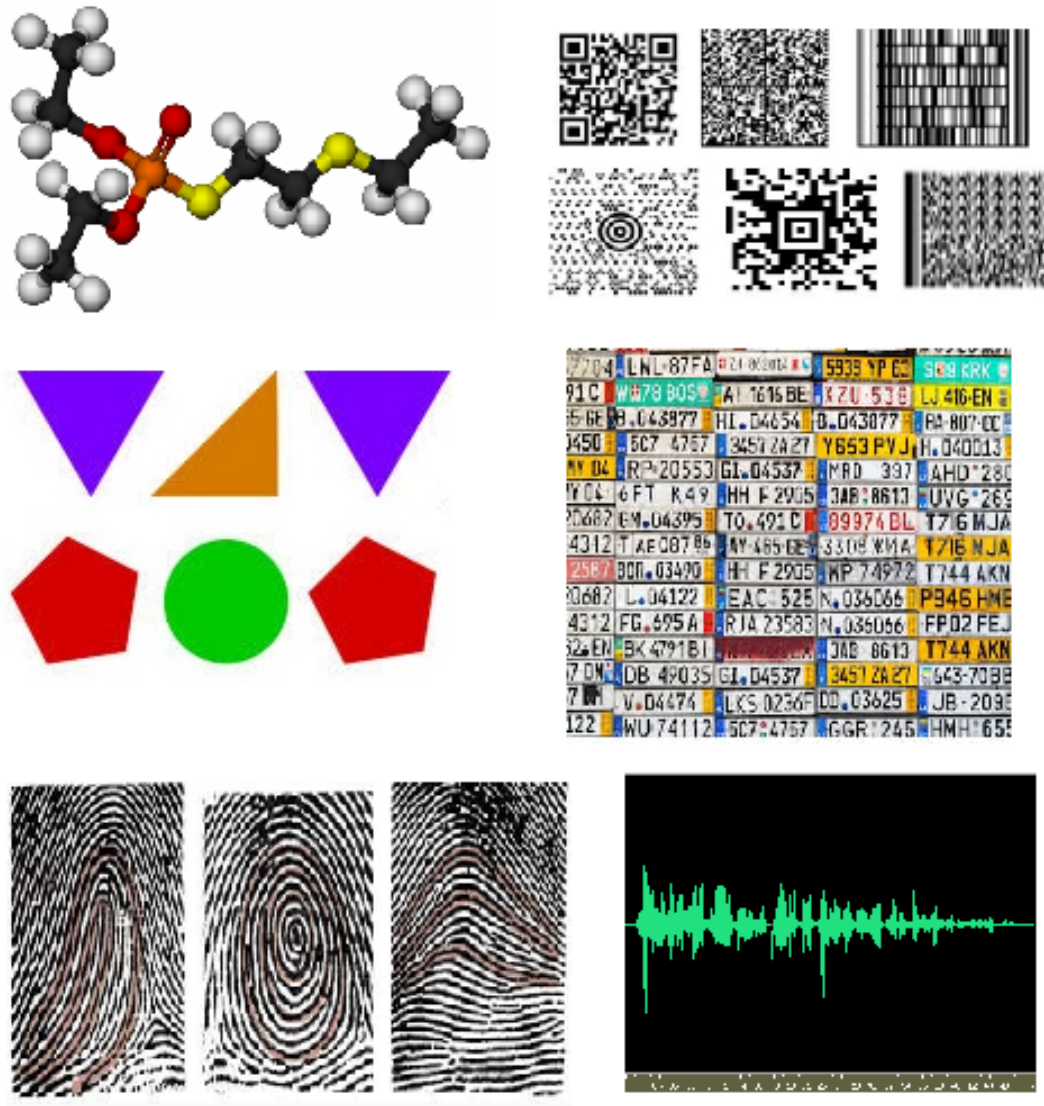


Figure 2.1: Different types of picture patterns

Pattern Recognition (PR) can be described as the scientific field of machine learning that focuses on the identification and classification of patterns into a number of categories or classes [Kpalma and Ronsin, 2007]. In this study, we focus on the description, classification and recognition of FA diagrams. According to Bishop [2006], the field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories. It involves acquiring raw patterns and taking actions (such as learning and classification) based on the category of the pattern to be recognised, as illustrated in Figure 2.2.

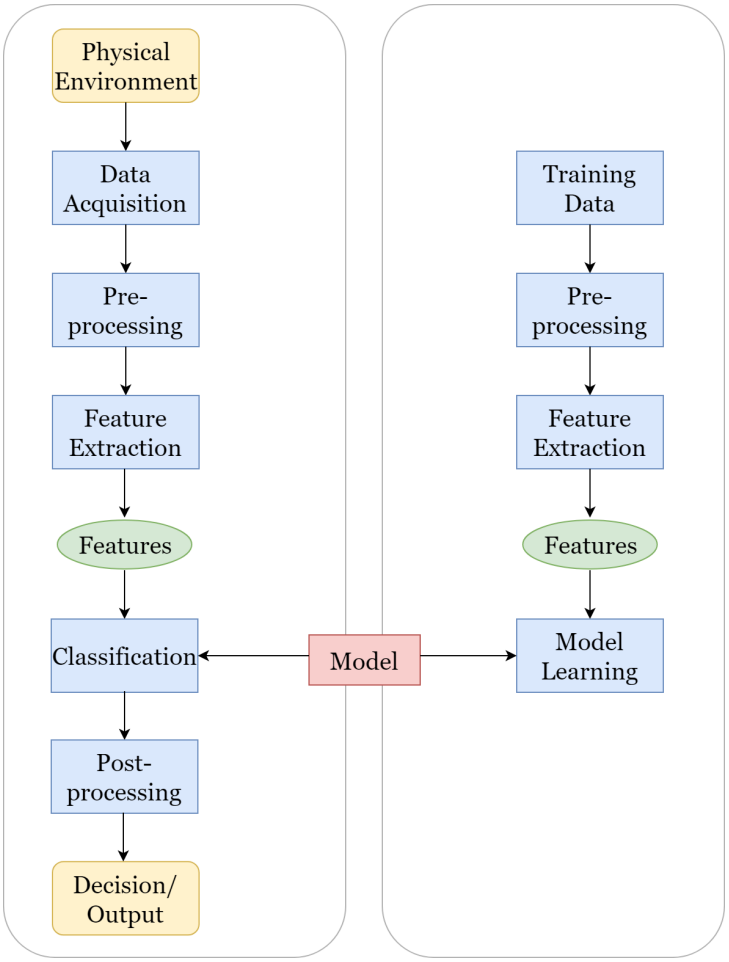


Figure 2.2: Components of a pattern recognition system [Essays, 2018]

The task of learning patterns can be done in two ways [Bishop, 2006] – supervised or unsupervised. During learning, if the given output or result datasets are used in training the machine to provide required outputs, it is said to be supervised learning whereas, in unsupervised learning, no datasets are given. Instead, the data is clustered into different categories. For example, suppose we are asked to arrange a bag of different fruits (i.e. apples, oranges, bananas) into the appropriate group. In supervised learning, this task will be performed easily because we already know what these fruits look like. We have been trained in the past to recognise these types of fruits. This makes the grouping easy. Now, in a situation where we have no clue of what these different fruits look like (i.e. we have never seen these fruits before) and we are given the task to arrange them in appropriate groups, firstly, we try to establish the physical similarities between these fruits (i.e. grouping them by their colours) after which we can create more groups by their shapes and so on. This principle is known as unsupervised learning, since the algorithm has no previous knowledge of the inputs, and it seeks to recognise patterns among the dataset.

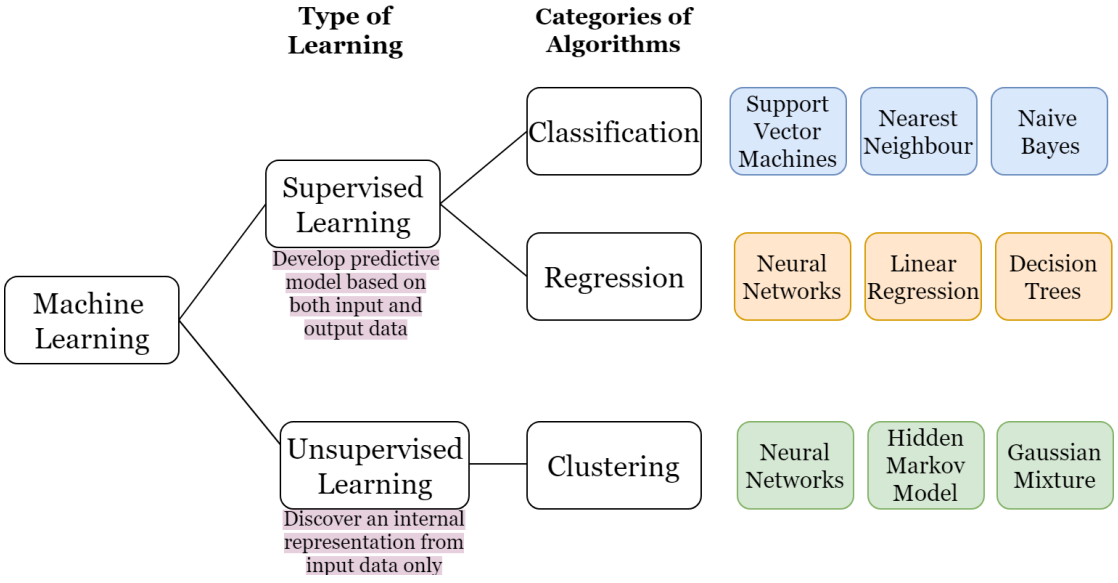


Figure 2.3: Type of learning

Most learning algorithms used in the past are based on supervised learning. Unsupervised learning, on the other hand, is not a well-explored field since it is challenging to define an unsupervised framework.

2.2.1 Supervised Learning

Supervised learning can be defined as when a learning algorithm is used to learn the mapping function $b = f(a)$ from an input variable (a) and an output variable (b). The aim of this learning is to estimate the mapping function well enough that when there is a new input (a), the algorithm can easily predict what the output (b) will be. According to Bishop [2006], algorithms in which the training data contains samples of the input vectors together with their corresponding target vectors are referred to as supervised learning. The learning process stops when the algorithm performance has reached an acceptable stage.

As shown in Figure 2.3, supervised learning algorithms can be divided into two problems, namely; *classification* and *regression*. The difference between these two problems is that the dependent attribute is categorical (category) for classification and numerical for regression. In this research, we focus on the classification problem of supervised learning techniques used in handwriting recognition. This is because we have created a pre-defined set of hand-drawn FA datasets and grouped the components of the FAs into categories or classes before saving the FAs in the created database. We then used the diagrams to train the recognition system. All the techniques used are explained in subsequent sections.

2.2.1.1 Classification

Classification is an essential and challenging technique used in biomedical engineering [Spanhol et al., 2015], remote sensing [Maggiori et al., 2016] and face recognition [Parkhi et al., 2015], to name a few. It is a supervised technique because it groups data based on prior information. A classification problem is when the output variable is a category, such as *dogs* or *cats*, *yellow* or *blue*. A classification model tries to make some assumption after observing some input values. For example, if a classification model is given one or more inputs, the model tries to predict the value of one or more outputs. To do this, a classification algorithm is used on the training data, and the extracted model is validated against labelled test data to evaluate the classification model accuracy and performance [Singh et al., 2016]. Classification can be either binary or multi-class classification.

Binary Classification: This is the process of classifying the objects of a given set into two groups using classification rules [Witten et al., 2016]. In this type of classification, the classification model predicts either yes or no; 0 or 1.

Multi-class Classification: This refers to the problem of classifying an object into one of three or more categories or classes [Witten et al., 2016].

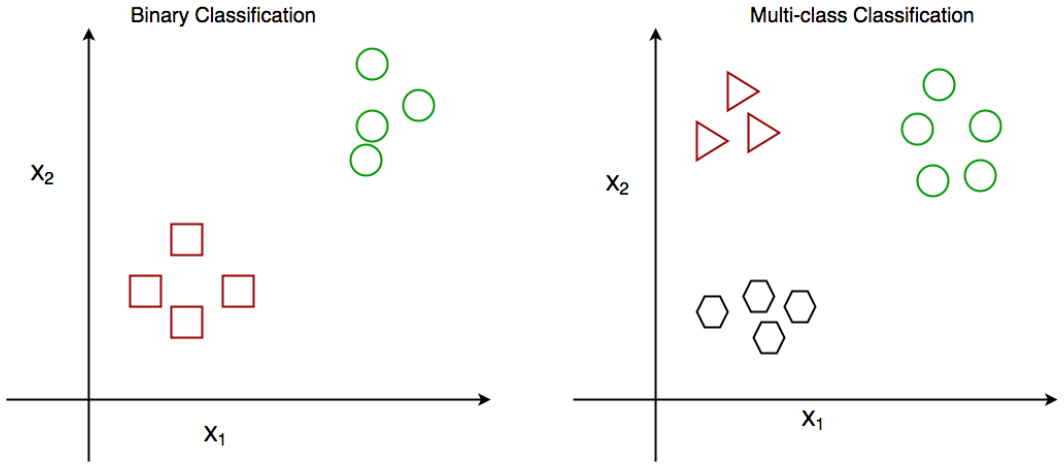


Figure 2.4: Binary classification and multi-class classification [Bisht, 2019]

In practice, many classification models have been developed mainly for binary classification; however, combining multiple binary classifiers can be used for multi-class classification. Classification models have been used in solving numerous problems. Examples of these problems are filtering of emails to determine if it is spam or not [Alghoul et al., 2018], differentiating between dogs and cats [Perez and Wang, 2017; Parkhi

et al., 2012], identifying dog breeds [Wang et al., 2014; Liu et al., 2012], and handwriting recognition [Yin et al., 2013; Poznanski and Wolf, 2016] (see Figure 2.5).

As said earlier, the focus of this research is the recognition of hand-drawn images and handwriting. More related work on image processing techniques used for handwriting classification and recognition will be explained in Section 2.6. Common algorithms used during classification include [Singh et al., 2016]: Naive Bayes, decision tree, random forest, gradient-boosted tree and one-vs-rest.

2.2.1.2 Regression

Regression is a supervised learning technique where the output has a real value, such as weight or salary. This technique is generally used when predicting the result of a given sample with its output values in the form of real values. This technique is mostly used to determine the cause and effect relationship between variables. It is also used in forecasting. Linear and logistic regression are the most common regression modelling algorithms.

Linear regression is a statistical model used when predicting the relationship that exists between dependent (Y) and independent variables (X) by looking at two factors. The red line in Figure 2.6 is called the best fit straight line. Using the available data, the best line that models the points is plotted. The following linear regression line equation can be used to model the problem.

$$Y = a + bX \qquad \text{(Equation 2.1)}$$

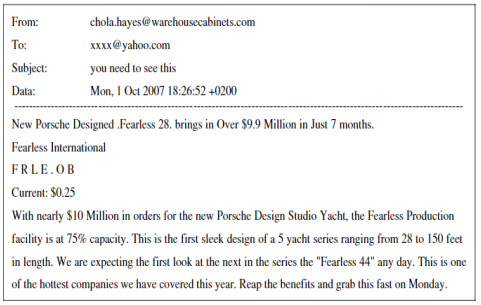
The main aim of linear regression is to find the best values for a and b, where a is the intercept (the value of y when $x = 0$) and b is the slope of the line.

2.2.2 Unsupervised Learning

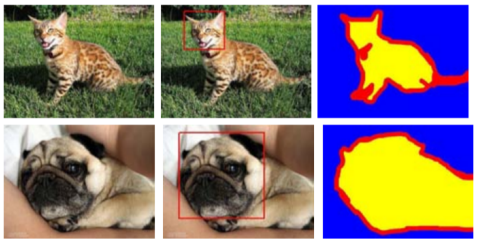
Unsupervised learning is a type of machine learning algorithm that uses information or data that is neither labelled nor classified to train a machine and allows the algorithm to make predictions with a minimum of human supervision. Unlike supervised learning, an unsupervised learning problem contains only the input variables x with no corresponding output variables. The underlying structure of the data is modelled using unlabelled data. An unsupervised learning algorithm typically makes up its own set of categories, using whatever features are available to it. This is done by taking a set of examples (with no category assignments) and classifying them into a set of groups



(a) Size-normalized examples from the Modified National Institute of Standards and Technology (MNIST) database [LeCun et al., 1998]



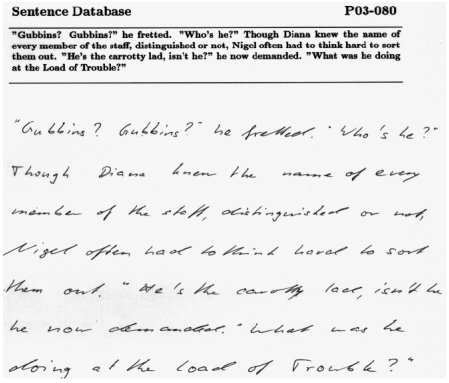
(b) Screenshot of a phishing email [Almomani et al., 2013]



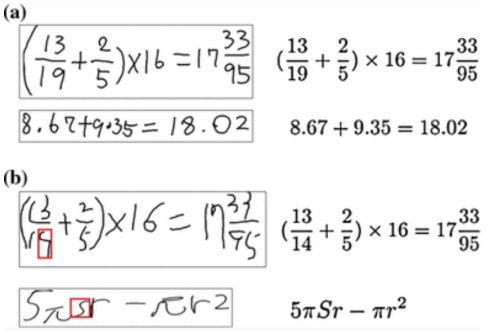
(c) Annotations in the Oxford-IIIT (International Institute of Information Technology, India) PET data. From left to right: pet image, head bounding box, and trimap segmentation (blue: background region; red: ambiguous region; yellow: foreground region) [Parkhi et al., 2012]



(d) Illustration of different dog breeds and their associated landmarks. The geometries associated with different breeds are discriminant. Within the same category, the shape constructed by the landmarks is similar [Wang et al., 2014]



(e) Handwriting from IAM-database [Gehrke et al., 2009]



(f) Examples in Hands-Math (recognition results are shown on the right of samples). (a) Correctly recognized mathematics expressions. (b) Misrecognised mathematics expressions (errors are located in red rectangles) [Le and Nakagawa, 2016]

Figure 2.5: Examples of images that have been used in classification problems

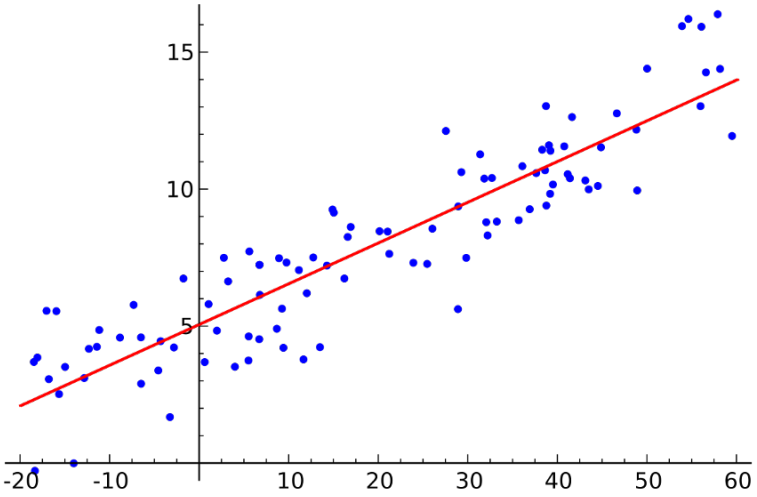


Figure 2.6: Linear regression [Gandhi, 2016]

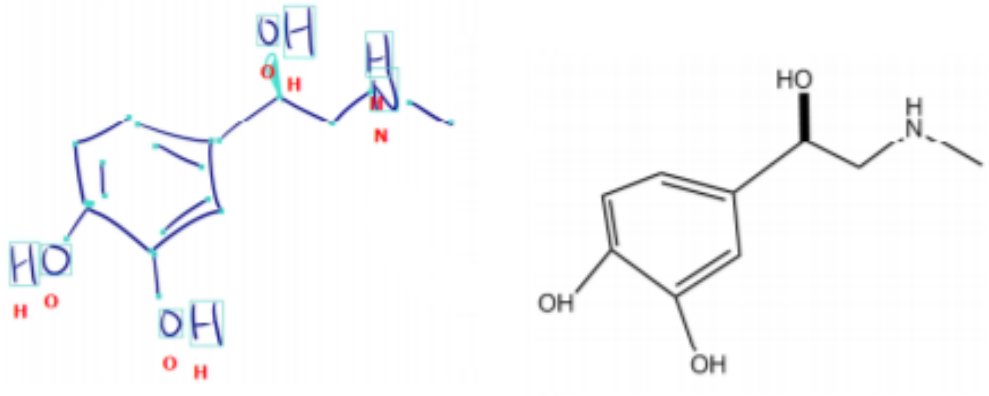
called clusters. The system tries to keep the members of any cluster similar to each other, usually in terms of a given distance metric. The most common type of this learning is clustering [Celebi and Aydin, 2016]. However, since unsupervised learning falls outside of the scope of this project, it will not be discussed in detail.

Clustering or cluster analysis is the most commonly used unsupervised learning technique. It involves the grouping of data points. It is used to classify a set of data points into their specific groups. Data points that are in the same group always have the same characteristics and properties, while those that belong to a different group must have a high number of distinct characteristics and properties. A distance metric is used to express the similarity between points. Clustering has been used in solving some image processing and computer vision problems such as face recognition [Schroff et al., 2015; Sankaranarayanan et al., 2016], object detection [Leichtle et al., 2017; Yuan et al., 2017], data mining [Schubert et al., 2015; Mirkin, 2016], speech recognition [Huo et al., 2015; Zhou et al., 2015] and text recognition [Yin et al., 2015; Wang et al., 2016].

2.3 Sketch Recognition

A notable application in the field of pattern recognition is sketch recognition. Researchers in fields that deal with hand-sketching such as software engineering, architecture, and engineering, are employing domain-dependent graphic notations for developing their work. Hand-drawn sketches are ubiquitous in our everyday life. Due to the flexibility and time-saving attributes of computers, most hand-written documents are now being transferred to digital documents and many domain-based sketch recognisers have been developed.

The aim of building sketch recognition systems is to allow people to convey graphical information in the same way that they have done in the past, i.e. simply by drawing. For example, a chemist could design the free-hand chemical diagram in Figure 2.7(a) and have the computer automatically construct the ChemDraw model in Figure 2.7(b). This process is what we mean by sketch recognition. In this thesis, we will show how it can be done by coming up with a system that can guide the extraction of FAs from hand-drawn sketches.



(a) An interpreted sketch with detected symbols highlighted

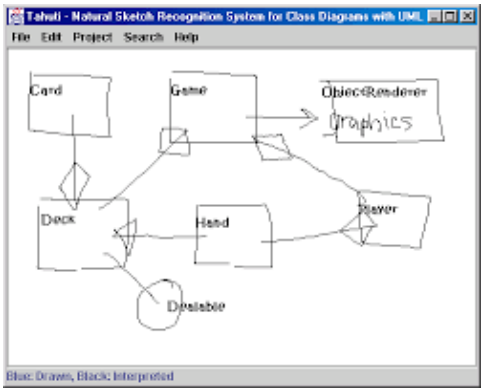
(b) The output structure generated in ChemDraw

Figure 2.7: Chemical structure interpretation process [Ouyang and Davis, 2011]

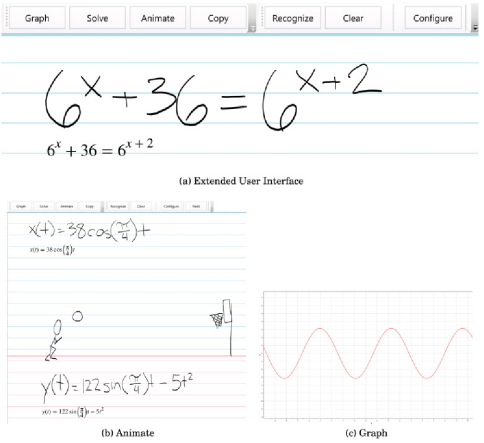
Sketch recognition can be described as the process of converting hand-drawn diagrams into symbolic computer representations, i.e. automated recognition of hand-drawn diagrams with a computer. The conversion process basically involves pre-processing and segmenting the hand-drawn diagram into a group of segments and giving a label to each class of components in the diagram. The label is then used to classify the components of the diagram. If computers can be programmed to detect and recognise these diagram sketches, then computers can provide an accurate and a regular (user-friendly) user interface for a large group of PC applications. These programmed systems can then help applications that utilise and make use of graphical representation of figures. Figure 2.8 illustrates such applications.

2.3.1 Sketch Recognition Techniques

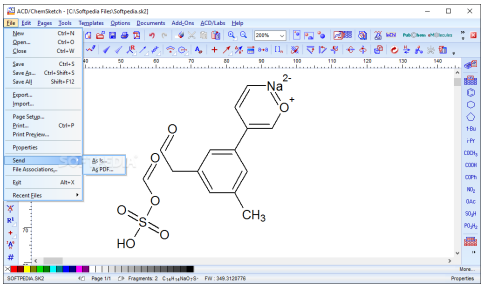
Sketch recognition is one of the many application areas of pattern recognition. During sketch recognition, an input image that is not known (i.e. input pattern) belongs to the category of one of the predefined symbols in a specific domain. The nature of the data determines patterns generated from the raw data [Kpalma and Ronsin, 2007]. The statistical feature of the data may be the basis of the generation of patterns. In some cases, the type of the pattern generated is determined by the underlying structure of the



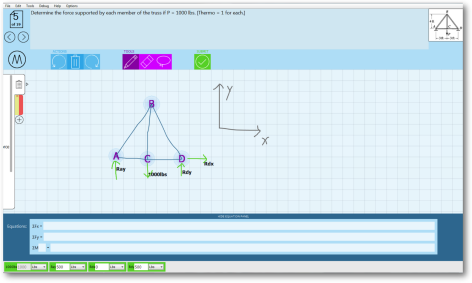
(a) UML class diagram recogniser [Hammond and Davis, 2006]



(b) A multiple recognition tool [Bott and Laviola Jr, 2015]



(c) Chemical structures modifier [Li et al., 2004]



(d) The Mechanix project [Nelligan et al., 2015]

Figure 2.8: Graphical representations of objects in some applications

data. In other cases, none of these data generation situations exists. In such scenarios, a system is developed and trained for desired responses. Thus, for a given problem, the solution may be obtained using one or more of these different methods. Therefore, there are various mathematical techniques that can be used to get the desired attributes for a recognition system. There are four well-known methods used for recognition. They are [Kpalma and Ronsin, 2007]:

(i) Template Matching

(ii) Statistical Classification

(iii) Syntactic Matching

(iv) Neural Networks

In template matching, there is a comparison of objects with stored examples or models that represent the underlying classes [Jain et al., 2000]. In a statistical approach,

random variables are used to explain patterns, from which class densities may be deduced. Modelling of data statistically is the basis of classification. The syntactic approach draws an analogy between the hierarchical structure of patterns and the syntax of languages. The simplest of a pattern sub-patterns are referred to as primitives. A complex pattern is represented by inter-relationships between these primitive patterns. Artificial neural networks attempt to solve practical pattern recognition problems by the application of the models of neural systems [Richardson et al., 2015].

Table 2.1: Pattern recognition models [Jain et al., 2000]

Approach	Representation	Recognition Function	Typical Criterion
Template matching	Pixels, samples, curves	Distance measure, correlation	Classification error
Statistical	Features	Discriminant function	Classification error
Structural or syntactic	Primitives	Grammar, rules	Acceptance error
Neural networks	Pixels, samples, curves	Network function	Mean square error

2.3.1.1 Template Matching

A high-level machine vision method which permits the identification of components of images that correspond to a given image pattern is called template matching (TM) [Kpalma and Ronsin, 2007]. It is a challenging and relevant area of computer vision, remote sensing, and computer security [Dekel et al., 2015]. Generally, there are two typical aspects involved in template matching: search strategy and similarity measurement [Grailu et al., 2009]. Due to template matching robustness, Normalized Cross-Correlation (NCC) is mostly used in similarity measurement [Krattenthaler et al., 1994]. Matching is done between entities like shapes of the same kind, points or curves to point out the similarities between them.

In template matching, a template or a model of the pattern to be recognised is made available. Considering all permitted operations like scale changes, rotation and translation, the template to undergo recognition is matched with the stored templates. There may be an optimisation of the similarity measure, mostly a correlation from the given training set. Here, the model image needs to be moved by the matching process to every available position in the source template and the similarity value computed accordingly. Therefore, the amount of time can be very high. Several global optimization techniques like genetic algorithms (GA) [Lee et al., 2001], particle swarm optimization (PSO) [Duan et al., 2010], and other hybrid learning techniques [Jain et al., 2001] have been used as a search strategy to minimize the amount of time.



Figure 2.9: (a) The templates are marked in green over the input images (b) The target images are marked with the detection result [Dekel et al., 2015]

Due to variation in handwriting, noise level and distortions in FA diagrams, template matching approach will not be suitable for the purpose of this study; because it will be difficult to select a good template from each pattern class as well as getting a proper matching standard will be a cumbersome task. Also, the correlation between an FA feature and an exactly matching region in the FA diagram may be less than the correlation between the feature and a bright spot. This will also make this approach unsuitable in this study.

2.3.1.2 Statistical Classification

The Statistical Pattern Recognition (SPR) approach assumes that the classification of data is based on statistics. This technique has been widely used in recognition of speech [Ho and Baird, 1997], damage detection in civil structures [Yao and Pakzad, 2012], bioinformatics (DeoxyriboNucleic Acid (DNA) sequencing) [Liew et al., 2005], and issues in other fields that need artificial cognitive help. An advantage of this technique is its capability of processing a large amount of information through continuous sensing. Moreover, this approach can easily be used in almost all applied science domains since it is a mostly mathematical tool.

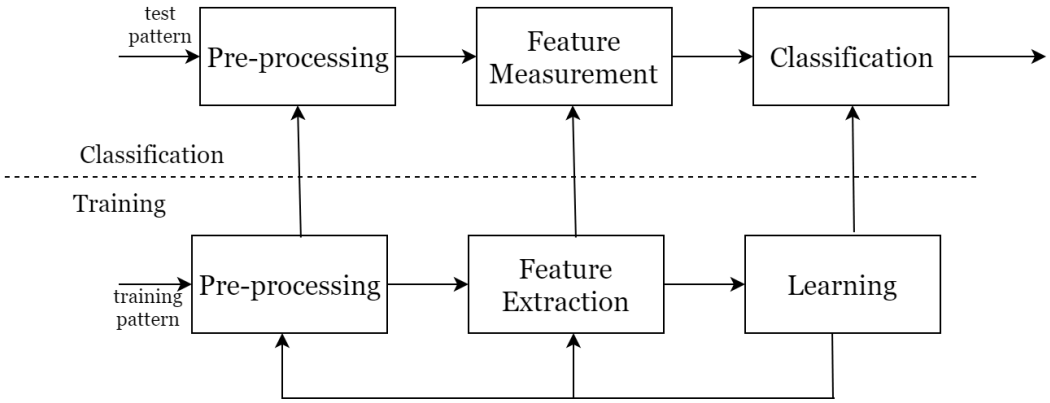


Figure 2.10: Statistical pattern recognition model [Jain et al., 2000]

In this approach, each pattern is represented in terms of d measurements or features and the pattern is viewed as a point in a d-dimensional space [Jain et al., 2000]. Classification is done in SPR by dividing the feature space into various classes. Hence, two things are important for classification performance: the techniques for partitioning the feature space selection and the feature selection. Two modes of operation are used in SPR: training (learning) and classification as shown in Figure 2.10. During the training, the module for extracting features looks for the suitable features to represent the data to be input to the system. A trained classifier will then be used for feature space partitioning. During classification, a classifier is trained to assign labels to data components.

In this research, FA diagrams acquired from students were used to train classifiers. One of these classifiers was used to separate the text from the graphics component of an FA. Also, a symbol classifier was introduced to classify all subsets of FA strokes. The classifier was trained to classify arrows as a symbol with a line and head. To classify an FA state symbol, the classifier was trained to classify a circle as a normal state and two concentric circles as a final state. The accuracies of these classifiers are reported in subsequent chapters.

2.3.1.3 Syntactic Matching

Syntactic matching is also known as structural pattern recognition. In this technique, complex structures that relate features to each other are used for recognition instead of the numerical feature vectors used in statistical pattern recognition systems [Venguerov and Cunningham, 1998]. The basis of syntactic matching is the primitives relation [Kpalma and Ronsin, 2007]. In this, patterns are seen in the form of a hierarchy that consists of simple structures composed by simpler structures. The simplest sub-structure or sub-pattern to be recognised is referred to as a primitive (see Table 2.1) and the given structure or pattern is represented as an inter-relationship between these primitives.

Different research has been done using this approach, most of which is based on a grammar representation of patterns or symbols [Roudaki et al., 2016; Strobbe et al., 2015; Costagliola et al., 2005; Mas et al., 2007]. According to Jain et al. [2000], a formal comparison is made between the structure of the input patterns and the syntax of a language. In doing this, primitives are viewed as alphabet letters that belong to a language, patterns are seen as a sentence in a language and the sentences are generated using a grammar. Hence, a large number of complex patterns can be described by using a few primitives and grammatical rules. As shown in Figure 2.11, a syntactic pattern recognition system can be described as consisting of three main parts; namely, pre-processing, pattern description or representation, and syntax analysis.

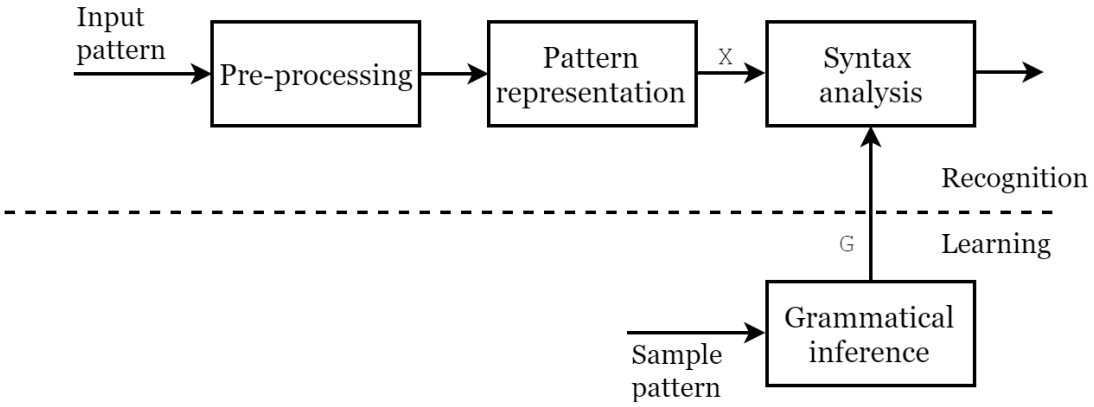


Figure 2.11: Syntactic pattern recognition model [Albus et al., 2012]

In contrast to statistical handwriting recognition and neural networks, syntactical handwriting recognition does not require a large amount of training data. This makes the success rate of this approach limited because of the ambiguity and the complexity in handwriting styles. Also, generating rules that are efficient for a large database of handwriting and automating the generation process is a difficult task. All this makes the syntactic approach not suitable for our research.

2.3.1.4 Neural Networks

Neural networks (NN) is a computational learning system that uses a network of functions to understand and translate a data input of one form into the desired output, usually in another form [DeepAI, 2019]. The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses. NN have been successfully applied in many tasks of pattern recognition and machine learning systems [Rastegari et al., 2016; Krizhevsky et al., 2012; Liang and Hu, 2015; Bengio et al., 2016]. The structure of the neural system is drawn from analogies with biological neural systems. Many algorithms have been developed to work with neural network learning [Reiher et al., 1996; Anquetil et al., 1999; Cesarini et al., 1997].

Generally, neural networks are made up of layers, as shown in Figure 2.12. These layers are composed of computational units known as neurons, with connections in different layers. The network processes the input data until it can classify it as an output.

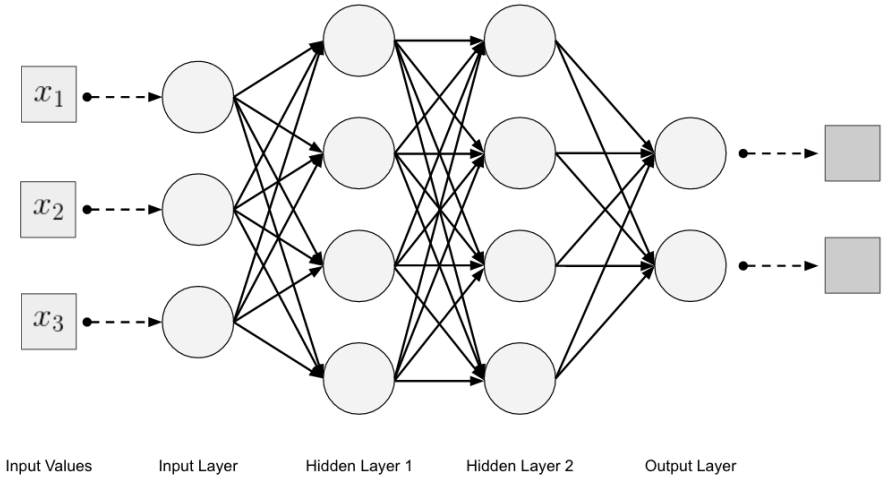


Figure 2.12: Neural network architecture [Patterson and Gibson, 2017]

2.4 Image Processing Techniques used in Sketch Recognition

Recognition of sketches is a complex task and numerous concepts have been used for various tasks involved in solving the problem of recognition. The frequent tasks, methods, tools and operations done in the field of diagram recognition and diagram analysis are described in this section. The concepts described in this section are mostly concepts suitable for the current research project and handwriting recognition in general. We did not go out of the scope of handwriting recognition in this section.

Pre-processing is an important step in building a recognition system. During document processing, the gathering of the data or images, in our case, is the first step. The process of scanning these images converts the image (a paper document) into an electronic document. During scanning, images might be affected by noise and these reduce the quality of the image and also affect the next step in the document processing. Therefore, the pre-processing stage is a crucial stage needed to remove all unwanted bits from the image and also improve the quality of the output image before using it as an input for the next processing stage. Scanned or captured diagrams can be grayscale, coloured or black and white.

2.4.1 Data Acquisition

Adequate and proper training, validation and test datasets are key requirements in developing a good recognition system, apart from using the right classification algorithm. Adequacy ensures that an adequate amount of data is available to learn the decision boundary as a functional mapping between the feature vectors and the correct class labels.

As explained in Section 1.1 and Figure 1.2, structured diagram recognition systems can be distinguished by the mode the system uses in acquiring its data. This can be done in two modes, namely; online and offline. Online systems input data [Ozawa et al., 2009] in an interactive manner from the system user (e.g. the user draws using electronic input devices such as tablets, electronic pens, etc.), while offline systems are referred to as the systems that perform image processing on scanned paper documents obtained offline from devices such as mobile phone cameras or scanners [Graves and Schmidhuber, 2009].

Research into creating practical computer recognition systems for online and offline recognition of structured diagrams is ongoing. There are still challenges in offline recognition research compared to online recognition. This can be because annotation and segmentation of images obtained using offline approaches is a more difficult task compared to online. Noise can be picked up during the acquisition process in offline data. If these noises are not dealt with properly, it will affect the final output of the recognition system.

2.4.2 Noise Removal

Noise removal is a field that has been addressed in machine printed or typed documents in document analysis. As explained in Section 2.4, digital images are liable to

different kinds of noise. Noise occurs as a result of errors that occur during the process of acquiring the data. Noise appears in an image as pixel values that do not reflect the true intensities of the real images or data. Noise can be introduced to an image through any of the following –

(i) Transmitting images electronically can introduce noise,

(ii) If an image is collected directly in a digital format, the mechanism and device for acquiring the image can introduce noise,

(iii) If the image is acquired with a scanner or a camera, the film grain is a source of noise. The scanner itself can also introduce noise.

There exist different types of noise that can occur during data acquisition; most of these have been addressed in Boyat and Joshi [2015]; He et al. [2010]; Yang and Yan [2000]; Maheswari and Radha [2010]. The most common noise that can occur when acquiring images like FAs is Gaussian noise and salt-and-pepper noise.

2.4.2.1 Gaussian Noise

Gaussian noise is a statistical noise with its probability density function (PDF) equal to that of the normal distribution, also referred to as the Gaussian distribution. In this type of noise, the values of the noise are Gaussian-distributed. The PDF p_G of a Gaussian random variable v is [Boyat and Joshi, 2015] –

$$p_G(v) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(v-\mu)^2}{2\sigma^2}} \qquad \text{(Equation 2.2)}$$

where v is the gray level of the image, μ is the mean value of each pixel of the image and σ represents the standard deviation. This type of noise occurs during image acquisition due to poor illumination, high temperature or transmission (electronic circuit noise). This noise can be removed by using spatial filtering, mean filtering and median filtering as explained in Section 4.6.

For hand-drawn FA diagrams, the connectivity and position of the transitions to other components (i.e. labels and states) have to be preserved to get a good result after the whole recognition process. During data acquisition, noise will be picked up from various sources when the diagrams are captured with either digital cameras or conventional film cameras [VenkateswaraRao et al., 2011]. When these diagrams are to be used in future, removal of the noise from the diagrams is required.

2.4.2.2 Salt-and-pepper Noise

A type of noise that can often occur in a document image during the process of converting the image to a digital format is salt-and-pepper noise, also known as impulse noise. It is also caused by dirt on the image [Farahmand et al., 2013]. The noise is made up of one or more pixels of random occurrences of both black and white intensity values. These pixels are assumed to be much smaller than the size of the original objects. When documents are printed, they appear in different forms. In such documents, salt-and-pepper noise appears like an inadequate ink in the document image. If the level of the noise is high, the segmentation, classification and recognition accuracy will be reduced. This type of noise can be removed by a median filter for diagrams such as FAs, flowcharts and UML diagrams.

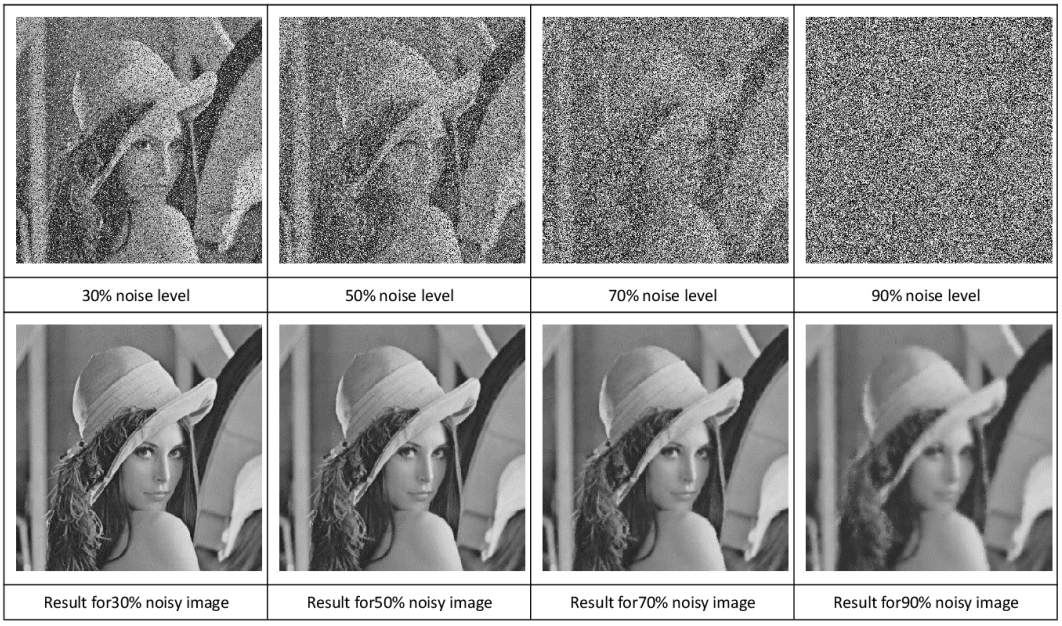


Figure 2.13: Images affected with salt-and-pepper and their corresponding output after filtering [Fu et al., 2019]

2.4.3 Filtering

Image analysis is usually simplified if the noise acquired during data gathering can be removed or filtered out. Filters are used during image processing to enhance the input image or image under consideration for further digital processing such as segmentation. During the filtering process, a pixel value of an image is changed, taking into account the pixel values of its neighbours.

2.4.3.1 Mean Filter

This kind of linear filter is implemented by a local averaging operation where the pixel value is replaced by the average of all the local neighbourhood values. A major weakness of the mean filter is its inability to preserve edges in an image.

2.4.3.2 Median filter

Median filtering is a non-linear operation [Sun and Neuvo, 1994]. In this operation, a grayscale image is used as input. The gray level of this image is replaced by the median of the pixels in the selected neighbourhood. This type of filtering technique works well on most types of noise. An advantage of this filtering technique is its ability to preserve image edges and preserve the image structures after removing the noise. The median is computed by sorting all the pixel values from the surrounding neighbourhood into a numerical order. Then the algorithm replaces the pixel under consideration with the middle pixel value of the sorted list. This filter is a sliding window spatial filter that replaces the middle pixel value in the window under consideration with the median pixel value. In this research, we used median filter to remove the noise from the FA images.

Consider Figure 2.14 with a 3×3 window of pixels of a noisy diagram. After applying median filter on the diagram, the noisy pixel 20 will be replaced by the median pixel value (see Figure 2.16). To calculate the median, the pixel values in the shaded 3×3 window in Figure 2.14 are first sorted into numerical order, before replacing the pixel under consideration with the median (middle) pixel value.

10	0	15	80	75	20
100	20	55	55	90	10
40	30	80	55	40	10
50	40	20	70	55	10
100	90	70	15	40	35
15	70	70	80	80	25

Figure 2.14: Image pixels affected by noise

The median filter processes a 3×3 window at a time before moving to the next 3×3 window. It stops its operation when all windows have been visited.

10	0	15
100	20	55
40	30	80

Figure 2.15: First 3×3 window of Figure 2.14

From Figure 2.15, we have pixel values;

$$10, 0, 15, 100, 20, 55, 40, 30, 80$$

The filter arranges the pixel values in an ordered form from the smallest to the largest number and picks the middle number as shown below. If the neighbourhood under consideration has an even number of pixels (e.g. a 2×2 or 4×4), the two middle pixels are averaged. The average value of these pixels is used.

$$0, 10, 15, 20, \mathbf{30}, 40, 55, 80, 100$$

For the above pixel values, the median pixel value of Figure 2.15 will be 30, i.e. Median = 30.

10	0	15	80	75	20
100	30	55	55	90	10
40	30	80	55	40	10
50	40	20	70	55	10
100	90	70	15	40	35
15	70	70	80	80	25

Figure 2.16: Pixels of Figure 2.14 after applying median filter on its first 3×3 window

2.4.4 Thresholding

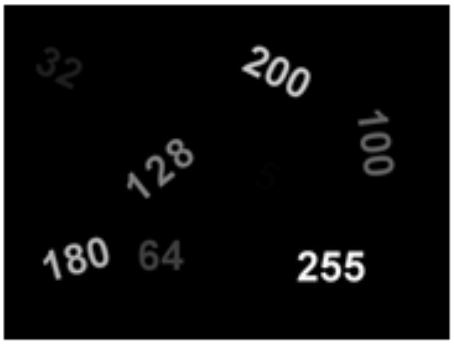
Thresholding an image involves setting all the pixels with values above the threshold to 1 and setting all pixels with values less than or equal to the threshold to 0. The output is a binary image in which the background is separated from the foreground [Birchfield, 2016]. Thresholding involves extracting the ink (foreground) from the paper (background) [Sonka et al., 2014]. The histogram of a grayscale image basically is composed of two peaks: a white background corresponding to a high peak (i.e. pixel brightness is greater than the threshold) and a black foreground corresponding to a lower peak (i.e. pixel brightness is lower than or equal to the threshold).

Basically, a threshold at any point in an image can be described based on the following: the function of the gray level of the image at the point, its neighbours, the gray level distribution of the image, and the spatial location of the point in the image under consideration. If the image is of X levels, then the threshold at the kth level is $0 < k < X$. This can be defined as,

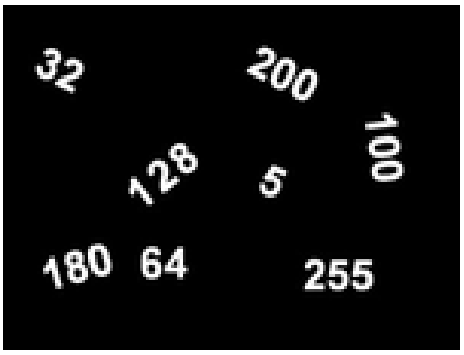
$$T_k(a, b) = f_k(g(a, b),\ n_k(a, b),\ p_n) \qquad \text{(Equation 2.3)}$$

where f_k represents the thresholding function for the kth level, a represents the row, b represents the column, $g(a, b)$ represents the gray level of the pixel at (a, b), $n_k(a, b)$ is the neighbours of pixel (a, b) and p_n represents the image gray level.

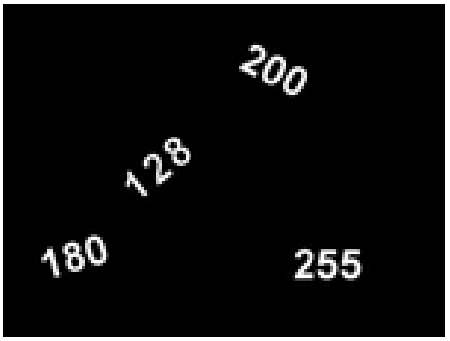
Image thresholding can be divided into global thresholding and local thresholding. Global thresholding is done when $T_1(a, b)$ is a function of p_n only, and this is done only once for the whole image. Local thresholding is done when the threshold $T_k(a, b)$ is a function of $g(a, b)$ and $n_k(a, b)$. In this research, global thresholding will be used. According to Babalola [2015], most diagrams like FAs and UML diagrams are presented and drawn with strokes (black) that have the same tone and are well differentiated from the paper background (white). Therefore, global thresholding using Otsu's method will be suitable for processing FA diagrams.



(a) Input image



(b) Threshold (thresh = 0 and maxValue = 255)



(c) Threshold (thresh = 127 and maxValue = 255)

Figure 2.17: Thresholding [Mallick, 2015]

For instance, let Figure 2.17(a) be the input image. In this figure, the background is black, while the foreground is white. This example is a simple illustration of thresholding and it is not based on Otsu's method. All the numbers in the figure are white, including 5, which is difficult to see because its grayscale value is 5. Note that all the numbers in Figure 2.17(a) have a grayscale value equal to the number itself (e.g. the

pixel value of the number 128 is 128, and that of 180 is 180). This makes 255 the brightest number and 5 the darkest. After applying binary thresholding on Figure 2.17(a) with the threshold set to 0 and the maximum value set to 255, we get Figure 2.17(b). Changing the threshold value to 127 removes all numbers less than or equal to 127 and produces Figure 2.17(c).

Otsu's method is one of the most used thresholding methods in the literature [Prasad et al., 2018; Sha et al., 2016; Mahmoud and Assayony, 2018]. Otsu's method is optimum because it maximises the within-class variance, a well-known technique used in statistical discriminant analysis. According to Carabias [2012], the within-class variance is the weighted sum of the variances of each class. This method chooses the threshold by minimising the within-class variance of the two groups of pixels separated by the thresholding operator. The aim of this method is to choose a threshold that minimises the combined spread [Carabias, 2012]. We used Otsu's method [Otsu, 1979] to threshold the hand-drawn FA diagrams because it is a general method and has proven to achieve state-of-the-art results.

Let the pixels of a given image be in L gray levels $[1, 2, ..., L]$. The number of pixels at the ith level is represented as a_i and $A = a_1 + a_2 + ... + a_L$ represents the total number of pixels. The gray level histogram is normalised and this is known as the probability distribution, presented as [Otsu, 1979]:

$$p_i = a_i/A, \quad p_i \geq 0, \quad \sum_{i=1}^{L} p_i = 1. \qquad \text{(Equation 2.4)}$$

To divide the pixels of an image into two classes C_0 and C_1 (i.e. background and foreground) by thresholding at level k, where C_0 represent pixels with levels $[1, ..., k]$ and C_1 represent pixels with levels $[k + 1, ..., L]$, we compute the probabilities of class occurrence and the class mean levels respectively as given below:

$$\alpha_0 = Pr(C_0) = \sum_{i=1}^{k} p_i = \alpha(k), \qquad \text{(Equation 2.5)}$$

$$\alpha_1 = Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - \alpha(k) \qquad \text{(Equation 2.6)}$$

and

$$\beta_0 = \sum_{i=1}^{k} iPr(i \mid C_0) = \sum_{i=1}^{k} ip_i/\alpha_0 = \beta(k)/\alpha(k), \qquad \text{(Equation 2.7)}$$

$$\beta_1 = \sum_{i=k+1}^{L} iPr(i \mid C_1) = \sum_{i=k+1}^{L} ip_i/\alpha_1 = \frac{\beta_\tau - \beta(k)}{1 - \alpha(k)}, \qquad \text{(Equation 2.8)}$$

where

$$\alpha(k) = \sum_{i=1}^{k} p_i \qquad \text{(Equation 2.9)}$$

and

$$\beta(k) = \sum_{i=1}^{k} ip_i, \qquad \text{(Equation 2.10)}$$

are the 0th and the first order cumulative moments of the histogram up to the kth level respectively, and

$$\beta_\tau = \beta(L) = \sum_{i=1}^{L} ip_i, \qquad \text{(Equation 2.11)}$$

represents the total mean level of the input image. The class variances are represented by

$$\sigma_0^2 = \sum_{i=1}^{k} (i - \beta_0)^2 Pr(i \mid C_0) = \sum_{i=1}^{k} (i - \beta_0)^2 p_i/\alpha_0, \quad \text{and} \qquad \text{(Equation 2.12)}$$

$$\sigma_1^2 = \sum_{i=k+1}^{L} (i - \beta_1)^2 Pr(i \mid C_1) = \sum_{i=k+1}^{L} (i - \beta_1)^2 p_i/\alpha_1. \qquad \text{(Equation 2.13)}$$

So, for each potential threshold, the algorithm separates the pixels into two clusters according to the value. Then, it finds the mean of each cluster and square the difference between the means. Finally, it multiplies the number of pixels in one cluster times the number in the other. This depends only on the difference between the means of the two clusters, thus avoiding having to calculate differences between individual intensities and the cluster means. The optimal threshold is the one that maximizes the between-class variance (or, conversely, minimizes the within-class variance).

Consider Figure 2.18(a), a 5×4 image with 8 gray levels. The pixels with gray level less than or equal to 3 are all assigned to the output label 0. All pixels with gray level greater than 3 are assigned to label 1 (see Figure 2.18(b)).

2	2	2	1	4
1	2	6	4	2
2	1	1	6	5
2	2	5	5	5

(a) A 5×4 image

$\xrightarrow{t=3}$

0	0	0	0	1
0	0	1	1	0
0	0	0	1	1
0	0	1	1	1

(b) Output of applying threshold $t = 3$ on Figure 2.18(a)

Figure 2.18: Thresholding a 5×4 image

Consider Figure 2.18(a) having N as its total number of pixels, i.e. $N = 20$, where n_i is the number of pixels at gray level i. The pixels in a given image is represented in L gray levels $(0, 1, ..., L)$. The normalised histogram value p_i can be calculated using Equation 2.14 [Xu et al., 2011].

$$p_i = \frac{n_i}{N}, \qquad \sum_{i=1}^{L} p_i = 1 \qquad \text{(Equation 2.14)}$$

Since there are four pixels with gray level value of 1 in Figure 2.18(a), $p_1 = \frac{4}{20} = 0.2$. This process continues for $p_2, p_3, ..., p_6$. The process stops at p_6 because it is the highest gray level value. The output of the processes above is illustrated in Figure 2.19.

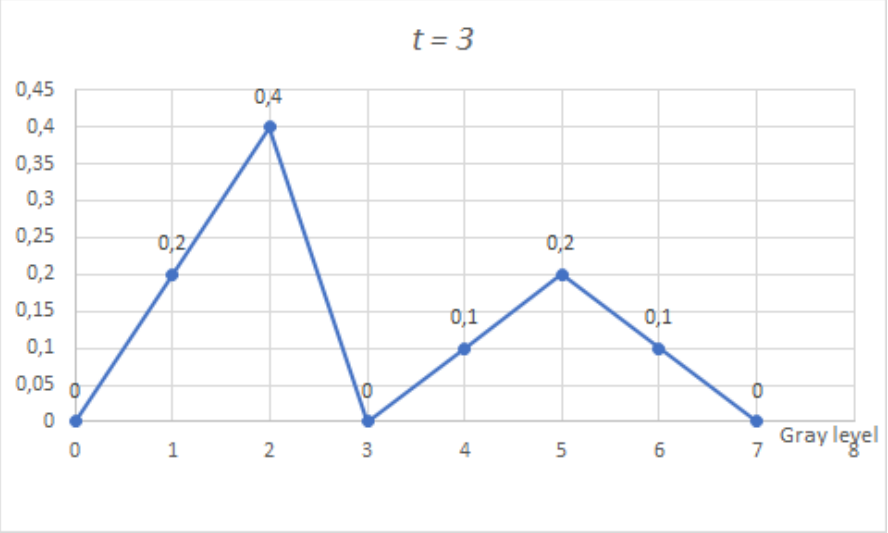


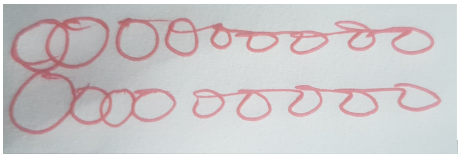
Figure 2.19: Normalised histogram of Figure 2.18(a)

2.4.5 Image Segmentation

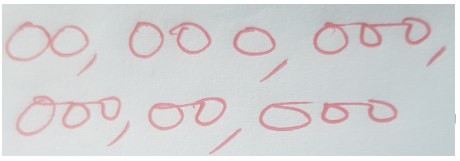
Image segmentation is a research area in image processing and computer vision that has gained much attention in recent time [Hu and Zanibbi, 2016; Ptak et al., 2017; Kovalevsky, 2019; Wang et al., 2019]. Image segmentation is the breaking down of an image into meaningful and simple segments (sets of pixels) for analysis [Chen et al., 2014]. Generally, it is an important step in image recognition, image analysis, object representation, and other image processing operations. Typically, this approach is used to find boundaries and objects (curves, dashes, and lines) in images. For the purpose of this research, we focus this section on segmentation of handwriting and hand-drawn image documents.

Segmentation of a document image into its basic components is regarded as a significant problem in the field of handwritten document recognition. The challenges that are encountered in handwritten documents makes the segmentation procedure a difficult process. Some of the major challenges include overlapping of image components, overlapping of words, an input image component been interpreted as noise, the difference in the skew angle between lines on the page or the same text line, the non-uniform spacing of words which occurs often, and adjacent text lines touching.

Handwriting touching and overlapping is a very complex challenge, not only because the writer has written the image components badly but also the way digits touch each other. This makes it hard to tell the number of characters in the touching string. For example, in Figure 2.20(a), it will be challenging for the computer to segment such image. This is understandable because even for a human, it will take a while to count the number of 0 string in the figure if there are more 0's in the figure than the one presented in Figure 2.20(a). To make segmentation and counting easy for humans, they add commas in large numbers like the one presented in Figure 2.20(b).



(a) A sample of handwritten touching digit of 0's



(b) A sample of handwritten touching digit of 0's separated by a comma

Figure 2.20: Text and Graphics Separation

Several works have been done in the field of handwriting segmentation [Messina and Louradour, 2015; Roy et al., 2016; Hu and Zanibbi, 2016]. Xu et al. [2014] proposed an over-segmentation approach with learning-based filtering that uses geometric features for single-touching Chinese handwriting. The proposed approach aims at separating

Chinese texts with low primitive segments, and also focus on finding the lines that separate these touching characters at a high recall rate. The approach starts by taking an input character string image. This image orientation is set to horizontal. Then the input image is segmented into connected components (CCs) [Kovalevsky, 2019] after setting its orientation. CCs that are highly overlapped are merged to form components. The output components with large width-to-height ratio or width are treated as candidate touching patterns. To know the width, the input image height is estimated as the mean of the height of every pair of consecutive components. A component is classified as candidate touching pattern if the width-to-height ratio is greater than θ_2 or if its width is greater than θ_1.

In Xu et al. [2014], θ_1 was empirically defined as 0.6 and θ_2 as 0.8. This was done to allow almost all the touching characters to be detected. Each of the detected characters is further processed in four stages, namely, construction of candidate separating lines, verification of separating lines by learning-based filtering, by heuristics, and post-processing. The candidate separating lines are generated using the contour information obtained before thinning and on the foreground of the thinned image. Redundant separating lines are removed using a learning-based filtering technique with geometric features. Projection analysis was used during the post-processing stage to separate the remaining components that are wide enough to generate extra separating lines. The splitting of the remaining under-segmentation errors on single-touching strings and the separation of multiple-touching and overlapping images were not addressed in this research. This makes it not suitable for the current studies as our dataset contains images with several overlaps and noise.

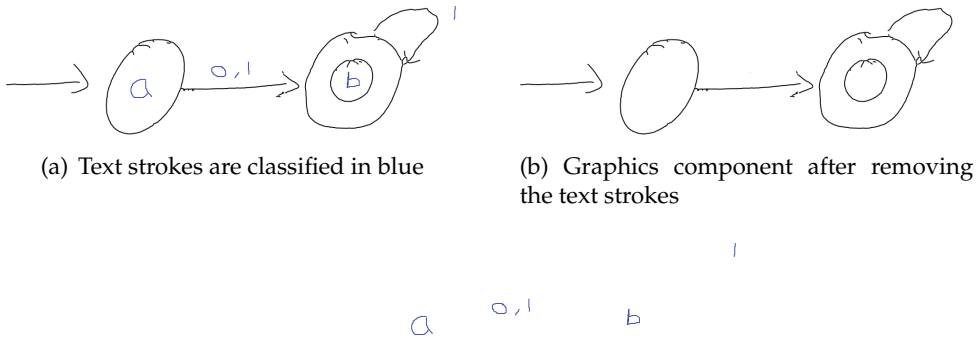
Connected bond analysis method was used in Tang et al. [2014] to segment handwritten chemical skeletal structural formulas. The method aimed to identify connected bond stroke (CBS) input and spilt it into many chemical single bonds. The method also finds the structural connection relationships that exist between the identified connected bond and chemical structures that are related to it. To split the CBS, it is important to find the connection points in the bond. These points are found at the corner of the bond strokes. These make the point identification problem to be computed as a corner detection problem. Given a CBS $S = \{p_0, p_1, p_2, ..., p_{n-1}\}$, the problem is to find a subset points $C = \{pC, pC \in S\}$ such that these stroke points are bond connection points in the CBS. The proposed connected bond analysis approach in Tang et al. [2014] consists of five stages, namely, bond connection point identification, false connection point detection, window size determination, bond splitting, and connection relationship identification. Temporal information of handwriting was leverage on in Tang et al. [2014]. However, due to the mode of our data acquisition, temporal information is not available. This makes the work proposed here not suitable for our research.

A simple notehead detector was proposed in Hajič Jr and Pecina [2017]. The proposed method was based on convolutional neural network and bounding boxes and was tested on the MUSCIMA++ dataset [Hajič and Pecina, 2017]. A notehead is an interface between the written score and music. Each notehead on a page represents one note to be played. This makes detection and recognition of noteheads an important step for optical music recognition. The detector proposed is made up of three components, namely, a target pixel generator which decides the region the detector should focus on, a detection network that jointly determines if the target pixel is part of a notehead and predicts it corresponding bounding box, and a filter that works on the combined predictions of the detection network and determines if the proposed bounding boxes truly matches to a notehead. The system of Hajič Jr and Pecina [2017] uses binary images. The network for detection was trained on augmented grayscale data. On average, the detector achieves a recall of 0.96 and precision of 0.97 on the used datasets. Despite the good result achieved, the runtime of their system is very slow with over a minute (i.e. 60 seconds) processing time for a single test image. This can be because image thinning was not performed on the images in the dataset. The filtering classifier is also fragile, i.e. its features are directly derived from the combined outputs of the detection network. Thus it is conditioned for a certain level of network performance.

During document processing, segmentation takes places in two stages. Firstly, text is separated from the segmented document containing both graphics and text contents, after which the graphics content of the document is broken down into individual classes [Rehman and Saba, 2014]. As discussed in Babalola [2015], when recognising an FA diagram, text-graphics segmentation (i.e. extraction of graphical content and textual information into separate layers) and symbol (graphics) segmentation are both required. Segmenting these diagrams involves two processing stages. The first segmentation process separates the graphical components and text in an FA into separate layers (see Figure 2.21). After this, another segmentation process takes the graphical part of the FA diagram and breaks it down into individual components.

2.4.6 Thinning

Thinning can be defined as an image processing operation that is used to remove some selected foreground pixels from binary images. During thinning, all lines in a diagram are reduced to the thickness of a single pixel. This operation is applied mostly to clean up the output of an edge detector by reducing all lines in the image to the width of a single pixel. The input and expected output of image thinning are always in the form of binary images.



(a) Text strokes are classified in blue

(b) Graphics component after removing the text strokes

(c) Textual component after removing the graphics strokes

Figure 2.21: Text and Graphics Separation

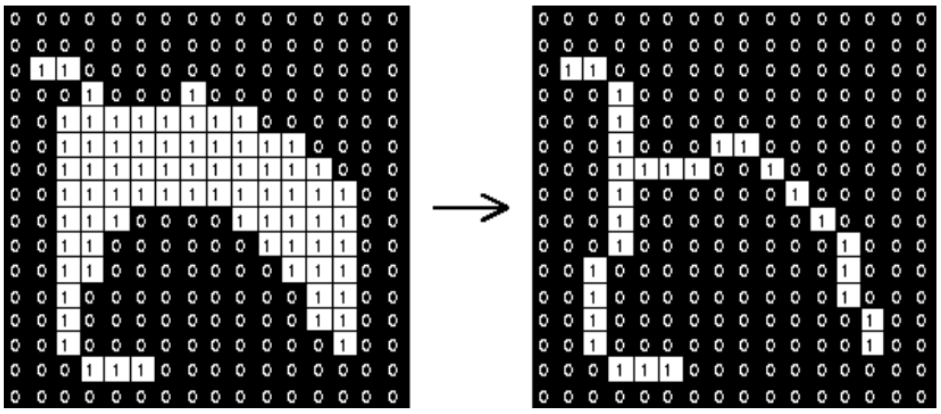


Figure 2.22: Thinning of a binary diagram [Robert et al., 2003]

The major reasons for image thinning are to;

(i) Simplify lines that make up the diagram into lines of a single-pixel thickness,

(ii) Reduce processing time,

(iii) Filter out unessential points in a diagram, and

(iv) Make data handling easy since all unessential points have been removed from the image foreground.

During the thinning process, the structural information of the diagram such as direction, position and length of lines must be retained. There exist different types of thinning algorithms in literature such as the Zhang Suen algorithm [Zhang and Suen, 1984], Guo and Hall's parallel thinning algorithm [Guo and Hall, 1989] and the edge-based thinning algorithm [Li et al., 2012]. Due to the simplicity and the wide use of the Zhang Suen algorithm, we initially adopted it in our research. During the experiment,

we noticed that the accuracy of the algorithm was low on some FA input diagrams and high on others. To have a consistent result and output, we improved on the existing Zhang Suen algorithm; this is explained later in Section 4.5.

2.4.6.1 Zhang-Suen Parallel Thinning Algorithm

The aim of the Zhang-Suen Parallel Thinning Algorithm is to remove all the contour points of an image except those points that belong to the skeleton, i.e. repeatedly remove or delete contour pixels that satisfy a number of conditions until a one-pixel wide 8-connected skeleton is achieved. It uses a 3×3 window as illustrated in Figure 2.23 and each element is connected to its eight neighbouring elements, i.e. their edges or corners touch.

A binarised diagram can be described as a matrix M, where each pixel $M(x, y)$ is either 0 or 1. The pattern of the diagram is made up of those pixels with the value 1. Extending this idea, the diagram can be represented as a large two-dimensional matrix of zeros and ones. Iterative transformations are applied on the matrix M point by point according to the values of the eight neighbouring points of the point. It is assumed that the neighbours of the point (x, y) are $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$, $(x-1, y-1)$, $(x-1, y+1)$, $(x+1, y-1)$ and $(x+1, y+1)$, as illustrated in Figure 2.23.

P_9 $(x-1, y-1)$	P_2 $(x-1, y)$	P_3 $(x-1, y+1)$
P_8 $(x, y-1)$	P_1 (x, y)	P_4 $(x, y+1)$
P_7 $(x+1, y-1)$	P_6 $(x+1, y)$	P_5 $(x+1, y+1)$

Figure 2.23: Nine pixels in a 3×3 window

The Zhang-Suen algorithm [Zhang and Suen, 1984] processes the black pixels with eight neighbours, leaving the pixels found at the corners and borders of the diagram out of the process. For the pixels to be analysed, Figure 2.23 shows the order of the pixels with P_1 representing the black pixel under consideration. In picture processing, the new value given to a point at the nth iteration depends on its own value as well as those of its eight neighbours at the $(n-1)$th iteration, so that all picture points can be processed simultaneously. It is assumed that a 3×3 window is used and that each element is connected with its eight neighbouring elements.

The Zhang-Suen process is divided into two sub-iterations.

Step 1: The aim of the first step is to locate the pixel P_1 that fulfils the five conditions listed below. If a pixel satisfies the five conditions, it will be set to white. All pixels are tested and pixels satisfying all the following conditions are noted at this stage.

(a) The value of the pixel in P_1 is 1 and it has eight neighbours (i.e. the pixel is not a corner or border pixel).

(b) The number of neighbouring black pixels for P_1 is at least 2 and not more than 6, i.e. $2 \leq I(P_1) \leq 6$, where $I(P_1)$ represents the number of black pixel neighbours of P_1, i.e. how many of the pixels from P_2 through P_9 are black.

(c) The number of white-to-black transitions around P_1 is equal to 1, i.e. $J(P_1) = 1$, where $J(P_1)$ is the number of transitions from white to black in the sequence $P_2,\ P_3,\ P_4,\ P_5,\ P_6,\ P_7,\ P_8,\ P_9,\ P_2$.

(d) At least one of $P_2,\ P_4,\ P_6$ is white, i.e. $P_2 \times P_4 \times P_6 = 0$.

(e) At least one of $P_4,\ P_6,\ P_8$ is white, i.e. $P_4 \times P_6 \times P_8 = 0$.

After iterating over the input diagram and identifying all pixels that satisfy the conditions in Step 1, all these pixels are then set to white. If any condition is not satisfied, the pixel remains black, i.e. P_1 will not be deleted from the diagram. For example, if the values of $P_2,\ P_3, ...,\ P_8,\ P_9$ are as illustrated in Figure 2.24, then $J(P_1) = 2$.

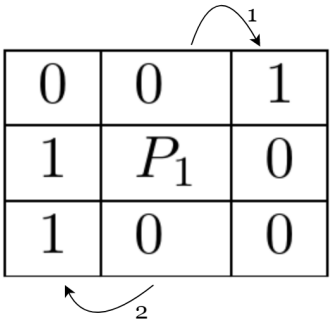


Figure 2.24: Counting the 01 patterns in the ordered sequence $P_2, P_3, ..., P_9$ [Zhang and Suen, 1984]

Step 2: All the pixels are tested again and pixels satisfying the conditions below are noted. The conditions are the same as for Step 1, except that the conditions in Step 1:d and Step 1:e above are changed as follows:

(a) For Step 1:d, $P_2 \times P_4 \times P_8 = 0$.

(b) For Step 1:e, $P_2 \times P_6 \times P_8 = 0$.

The rest remains the same. Again, all the pixels of P_1 satisfying the conditions in Step 2 are set to white, but only after the pixels meeting the conditions have been noted. The two steps above are iterative, i.e. Step 1 followed by Step 2. When no points fulfilling the requirements remain, the remaining points form the regional skeleton.

2.4.6.2 Thinning with Structuring Elements

During thinning, structuring elements (see Figure 2.26) are used to probe the image under consideration. It is a sub-image used to probe an image under consideration or probe for features of interest. Basically, the structuring element is a binary matrix where pixels with value 0 are not active and pixels with value 1 are active. When a structuring element is placed over a binary image, the three possible scenarios listed below and illustrated in Figure 2.25 can occur:

Fit: It entirely overlaps the image foreground, i.e. for each pixel set to 1, the corresponding pixel in the image is also 1.

Hit: A structuring element is said to hit an image if it partially overlaps the image foreground, i.e. for at least one of its pixels but not for all, the corresponding image pixel is also 1.

Miss: A structuring element is said to miss an image if none of its pixels overlaps the image foreground.

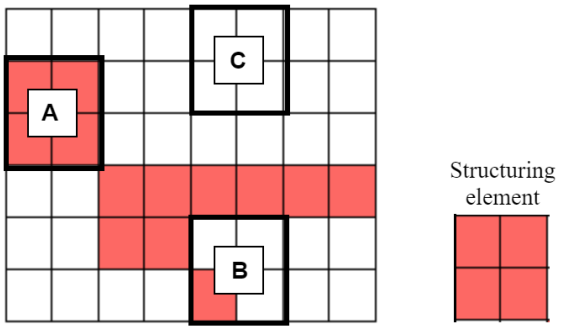


Figure 2.25: Hovering over an image with structuring elements

In Figure 2.25, the structuring element fits the image in A and hits or intersects the image in B and misses the image in C. The structuring elements are in the order $B = (B_1, B_2, ..., B_n)$. Figure 2.26 illustrates the structuring elements used during image thinning. Each of the elements depends on each other element, i.e. to get B_2, B_1 must be rotated by one pixel in a clockwise direction. This approach is applied to each of the structuring elements to get the structuring element next to it. For the elements

in Figure 2.26, the pixels with label x are not considered (i.e. they are ignored during the thinning process) when applying the structuring elements to an image because the aim of the process is on getting a hit for any central pixel in an image with value 1. We focus on identifying a match between the structuring element and the input image when the structuring element is placed over the input diagram. If a match is found, we say there is a complete match. Then the central pixel with value 1 will be changed to 0. Applying the structuring elements on an image makes the application of the thinning method rules on the image easier.

0	0	0
x	1	x
1	1	1

(a) B_1

x	0	0
1	1	0
1	1	x

(b) B_2

1	x	0
1	1	0
1	x	0

(c) B_3

1	1	x
1	1	0
x	0	0

(d) B_4

1	1	1
x	1	x
0	0	0

(e) B_5

x	1	1
0	1	1
0	0	x

(f) B_6

0	x	1
0	1	1
0	x	1

(g) B_7

0	0	x
0	1	1
x	1	1

(h) B_8

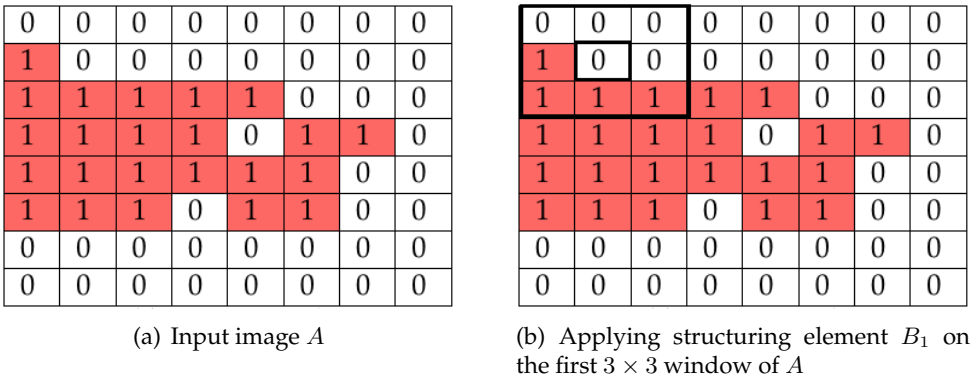
Figure 2.26: Sequence of rotated structuring elements used for thinning

The process and results of applying the structuring elements to the input diagram (i.e. Figure 2.27(a)) are explained below.

Note that for the images in the explanation below, the 3×3 window under consideration is the one highlighted in the black border. Each of the structuring element is placed over the highlighted region in the input image with a focus on finding a hit on the central pixel.

- In Figure 2.27, the input image A (see Figure 2.27(a)) is a binary 8×8 image to be thinned. The shaded pixels with value 1 are the foreground (ink) while those with value 0 are the background pixels. To get the area of interest or the area that fulfils the skeleton condition in A, structuring elements are placed over each 3×3 window in A. The first structuring element to be used on A is the element B_1. The operation starts at the first (top-left) 3×3 window of A (see Figure 2.27(b))

and moves to the next 3×3 window in A as illustrated in Figure 2.27(c). The aim of the operation is to find a central pixel hit. Once a hit is found, a new image is produced. This process is continued until B_1 is applied to the last 3×3 window in A. Once the process is completed, the element B_2 is then applied to the output of the B_1 process.



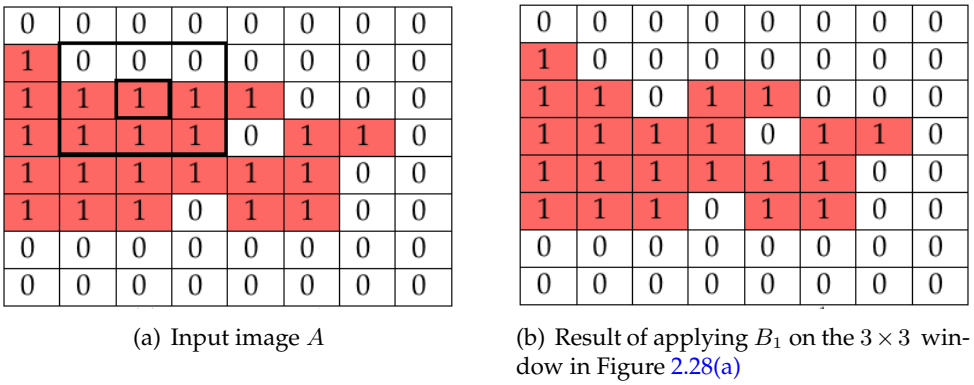
(a) Input image A

(b) Applying structuring element B_1 on the first 3×3 window of A

(c) Applying structuring element B_1 on the second 3×3 window of A

Figure 2.27: Applying B_1 on input image A

- Figure 2.28(b) shows the output A_1 of applying B_1 on A.



(a) Input image A

(b) Result of applying B_1 on the 3×3 window in Figure 2.28(a)

Figure 2.28: Output of applying element B_1 on input image A

- The process of applying element B_2 on the output image A_1 starts from the first 3×3 window in A_1. The structuring element is used to probe all the 3×3 windows

in the image A_1. Applying the structuring element B_2 on A_1, the output image A_2 of the process is illustrated in Figure 2.29. Only a single hit is found in this operation after probing the images with the structuring elements.

0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	0	0
1	1	1	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a) Applying B_2 on A_1

0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	0	0
1	1	1	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) Result of applying B_2 on each of the 3×3 windows in Figure 2.29(a)

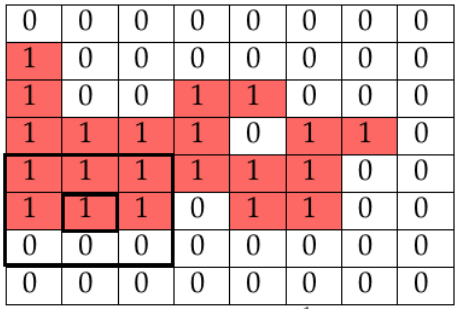
Figure 2.29: Applying B_2 on input image A_1

- Similar to what has been done for elements B_1 and B_2, image A_2 is used as the image under consideration. Applying elements B_3 and B_4 on A_2, will result in a miss. All the 3×3 elements in this image were probed using the two elements i.e. B_3 and B_4. They did not produce a new image.
- Figure 2.30 shows the input images A_2 and A_3 and their corresponding outputs A_3 and A_4 respectively after probing the input images (A_2 and A_3) with element B_5.
- Applying elements B_7 and B_8 will have no effect on the input image.

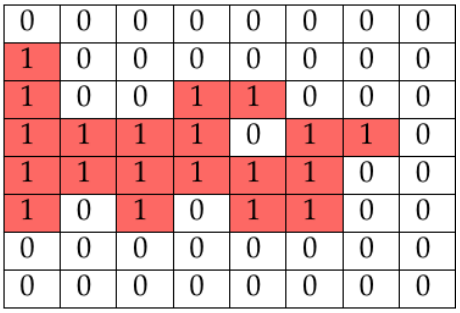
The final output of applying the eight structuring elements in Figure 2.26 on input image A is shown in Figure 2.31(b). The image is thinned; however, the pixels at the edge of the image were not considered during thinning. This makes the output not perfectly suitable for further processing.

2.4.7 Chaincode

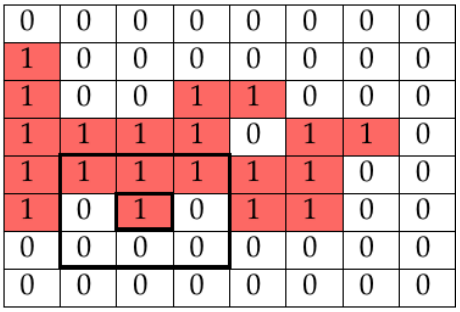
Chaincode is a type of boundary representation technique used for regional boundaries and geometry curves in image processing, and computer vision. It requires a small amount of storage to store large information. Chaincode can be used to illustrate any closed borders or curves [Lan-yi and Shu-guang, 2013]. Generally, chaincode can be divided into two, namely; Freeman chaincode in four directions and eight directions.



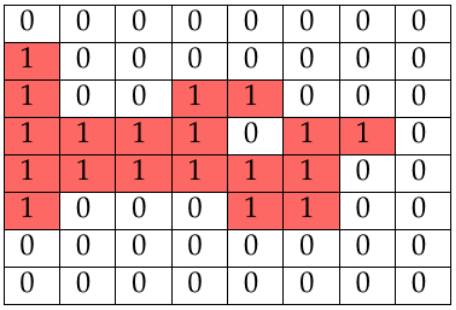
(a) Applying B_5 on A_2



(b) Result of B_5 on each of the 3×3 windows in Figure 2.30(a)

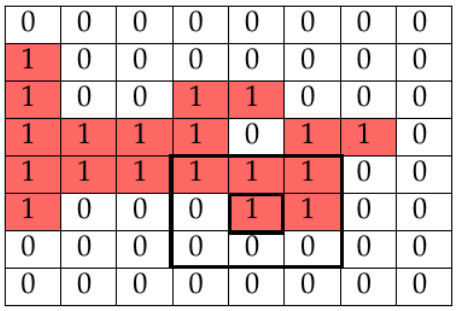


(c) Applying B_5 on A_3

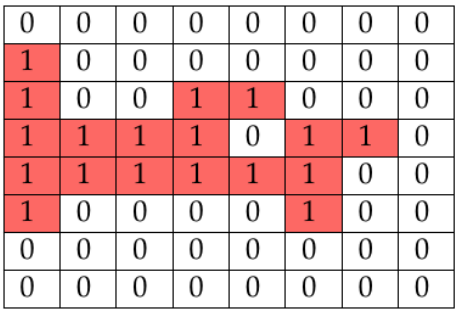


(d) Result of applying B_5 on the 3×3 windows in Figure 2.30(c)

Figure 2.30: Applying B_5 on input images A_3 and A_4



(a) Applying B_6 on A_4



(b) Result of applying B_6 on the 3×3 windows in Figure 2.31(a)

Figure 2.31: Applying B_6 on input image A_4

The calculation of the perimeter is given by [Fu and Han, 2012]:

$$\text{Perimeter} = \sum_{i=1}^{n} a_i, \qquad \text{(Equation 2.15)}$$

where a_i is the chaincode of the curve and the perimeter is composed of $P_0, P_1, ..., P_{n-1}$. For the chaincode in four directions, a_i is represented by;

$$a_i = \overline{P_i P_{i-1}} = 1, i = 0, 1, 2, 3. \qquad \text{(Equation 2.16)}$$

For the chaincode in eight directions, a_i is given by;

$$a_i = \overline{P_i P_{i-1}} = 1 : i = 0, 2, 4, 6; \sqrt{2} : i = 1, 3, 5, 7. \quad \text{(Equation 2.17)}$$

The perimeter is made up of $P_0, P_1, ..., P_{n-1}$, which are a total of n segments between $n + 1$ boundary points. The area is given by;

$$\text{area} = \sum_{i=1}^{n} a_{ix}(y_{i-1}\frac{1}{2}a_{iy}), \quad \text{(Equation 2.18)}$$

$$y_i = \sum_{j=i}^{i} a_{jy} + y_0, \quad \text{(Equation 2.19)}$$

where n represents the number of the chaincode, a_{ix} and a_{iy} are respectively x and y components of the direction chain (i.e. four or eight), y_i is the coordinate of y for each point and y_0 is the coordinate of y for the starting point. During the extraction of circles in our research, we used an approach similar to the Chaincode approach, i.e. we used the direction of the circle shape pixel connectively.

2.5 Application Domain of Sketch Recognition

As explained earlier, various fields including remote sensing, artificial intelligence, document classification, surveillance, medical imaging, bioinformatics and psychology have used sketch recognition. It has helped in solving numerous problems like image retrieval, optical character recognition (OCR) and biometric authentication. Alvarado and Davis [2004] discussed most of these issues in their research and elaborated on some of the key concepts involved in sketch recognition.

In this section, we focus on existing recognition systems designed for diagrams that have common structural and visual features or properties (like arcs, lines, circles, arrows and states) with FA diagrams. Basically, FA diagrams are made up of simple shapes as illustrated in Figure 1.3. As discussed in Section 1.1, research of online recognition systems has been the focus of more studies when compared to offline recognition systems. Hence, most of the systems described in this section use the online approach during recognition. Since most of the recognition systems described in this section are based on online recognition, it is expected that these systems should be quite simple to develop compared to offline systems. Because the data or images used in online systems contain little or no irregularities, noise removal and thinning of the images

may not be needed. Also, understanding the relationship that exists between the components of a diagram may not be needed in online systems and having to deal with variability in handwriting and handwriting style from person to person during segmentation may not be necessary. However, in this study, all the problems stated above are addressed since the focus of our research is on offline recognition system.

2.5.1 Unified Modelling Language

The Unified Modelling Language (UML) is a standard language used in object-oriented system modelling [Evans et al., 1998]. UML represents a diagram notation developed to model big object-oriented systems. Tools used in modelling these systems are called Computer-Aided Software Engineering (CASE). A large amount of research has been done in this field [Karasneh and Chaudron, 2013; Ho-Quang et al., 2014].

Lank et al. [2000] achieved a state-of-the-art result in their experiment where they designed and implemented an online system (see Figure 2.32) used for recognising hand-drawn UML diagrams. In this system, a mouse and an electric board are used to produce input. To create an input, the users create a diagram by drawing symbols on a software drawpad. The drawpad collects a set of points which are grouped into strokes by using a mouse. These pen-strokes are then grouped into symbols by a sophisticated segmentation algorithm, adapting to drawing imprecisions that are usually seen in hand-drawn input. The organisation of the system is around a retargetable kernel by which a user interface is presented to the user for online recognition of any UML notation. The retargetable segmenter algorithm receives a set of strokes from the drawing pad and groups them in strokes. After the initial recognition by the algorithm, characters and symbols that are recognised are sent to the another segmenter (i.e. the UML segmenter) for UML specific segmentation and recognition. The process of passing data between the two segmentation processes is done by a filter. The filter helps to convert the grouped strokes into the required data format as well as pass the data to the appropriate recognisers. The approach described here is only for UML diagrams; hence it cannot be used for other structured diagrams. Issues that involve the recognition of the logical relationship that exists between diagram symbols and the meaning a group of symbols is communicating were not addressed in their work. It is important for recognition systems to provide the meaning of the relationships that exist between the symbols of a diagram and also provide a meaning for the entire diagram. Hence, these issues will be addressed in our research. We explored the relationships that exist between FA diagram components during extraction, segmentation and recognition of FA diagrams.

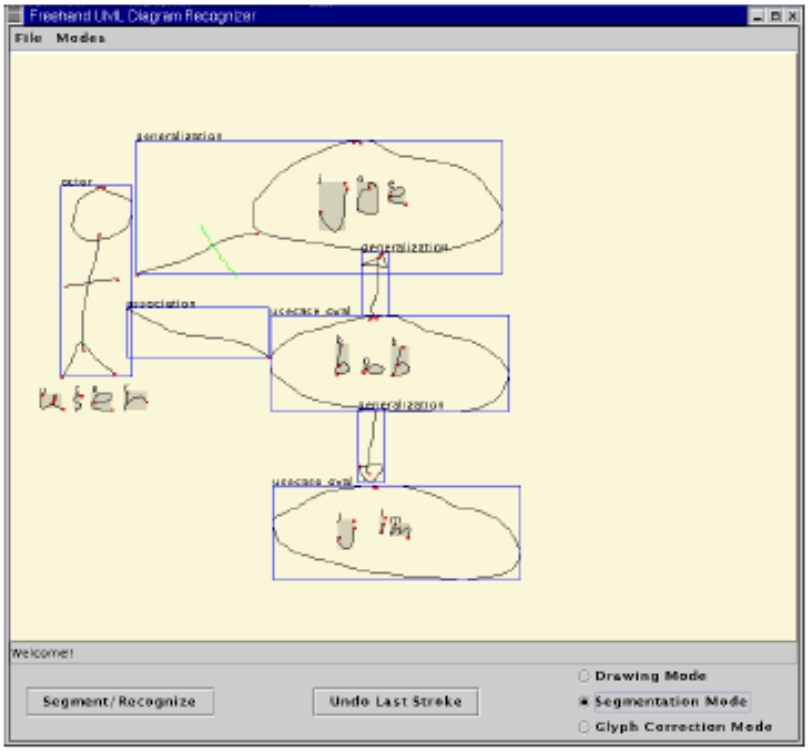


Figure 2.32: Hand-drawn UML diagram recogniser [Lank et al., 2000]

2.5.2 Flowchart

A flowchart is a diagram generally used in representing process flow or algorithms by using a set of symbols and flow lines. Often, designers make use of commercial software like Microsoft Visio [Halpin et al., 2003] when drawing a flowchart. Flowcharts can be drawn with Visio by dragging and dropping the components of a flowchart on the computer screen. However, this method has some setbacks [Miyao and Maruyama, 2012], e.g. the drawing area is or might be reduced by the menu area and the symbol icon area. Also, navigating the mouse across all these areas might be tiring and time-consuming. This prompted the development of flowchart recognition systems [Awal et al., 2011; Yuan et al., 2008; Bresler et al., 2014].

Chen et al. [2015b] developed a flowchart recognition system based on graph grammars and logical structure. The system is made up of the following stages: firstly, the strokes were grouped based on temporal and spatial information; dynamic programming was used to determine the stroke combination. Image segmentation was also performed using dynamic programming. In their work, they defined two additional features for a flowchart diagram during implementation; namely, a symbol is made up of certain numbers of strokes, and the index strokes in a symbol are not always consecutive. Secondly, a combination of the strokes was classified using WEKA [Hall et al.,

2009]. The classification algorithm used was the random forest algorithm [Liaw et al., 2002]. The classifier used 23 dominion feature values introduced in Blagojevic et al. [2011] as input signals. The used feature values mainly consist of direction, curvature, point of intersection and density. Thirdly, the flowchart was analysed and the graph grammar was constructed. Lastly, the graph grammar was parsed and a final decision was made. The grammar was parsed to get a global logical solution. Constraints such as irregularities in drawing, and drawing styles were not considered in this work. The accuracy of the classifier they used was quite low as it misclassified 23.2% of the symbols presented to it. During our classification phase, it is important to reduce the error to the bare minimum in order to get a good result during recognition.

2.5.3 Graph Diagram

Graph diagrams basically consist of nodes (circles) and their interconnections (lines). A study by Auer et al. [2013] used optical graph recognition (OGR), i.e. converts a digital image of a drawn graph into its topological shape (see Figure 2.33). The input to the system is a digital drawing of a graph and the expected output is a description of the graph in a GraphML file format [Team, 2015]. Their approach can be broken down into four stages: pre-processing, segmentation, topology recognition, and post-processing. The pre-processing stage separates the pixels of the drawn graph from the background pixels. The image is also binarised and enhanced by using a morphological noise reduction technique. The segmentation stage finds the vertices in the binary image after pre-processing. In particular, it determines whether an object pixel belongs to an edge or a vertex. The output of the segmentation stage is a ternary image that has three colours for the edge, vertex and background pixels. The third stage, topology recognition, was the main focus of their work. It includes skeletonization (for obtaining a 4-connected skeleton using morphological thinning), edge classification (groups the pixels of the skeleton using their 4-neighbourhoods) and recognition of pixels as vertex pixels. The output of this stage are crossing pixels, edge pixels, port pixels. vertex pixels and miscellaneous pixels. An advantage of the pixels classifier used is that it allows users to identify edge sections in the image. The post-processing stage completes the recognition process. This stage consists of procedures that use the topological structure as input. The recogniser was evaluated using a dataset of graph images. This dataset consists of 92,272 graph drawings and was obtained from 11,534 Rome graphs where each of the graphs was drawn eight times using a spring embedder algorithm. The recogniser achieved an accuracy of 89.68% during evaluation.

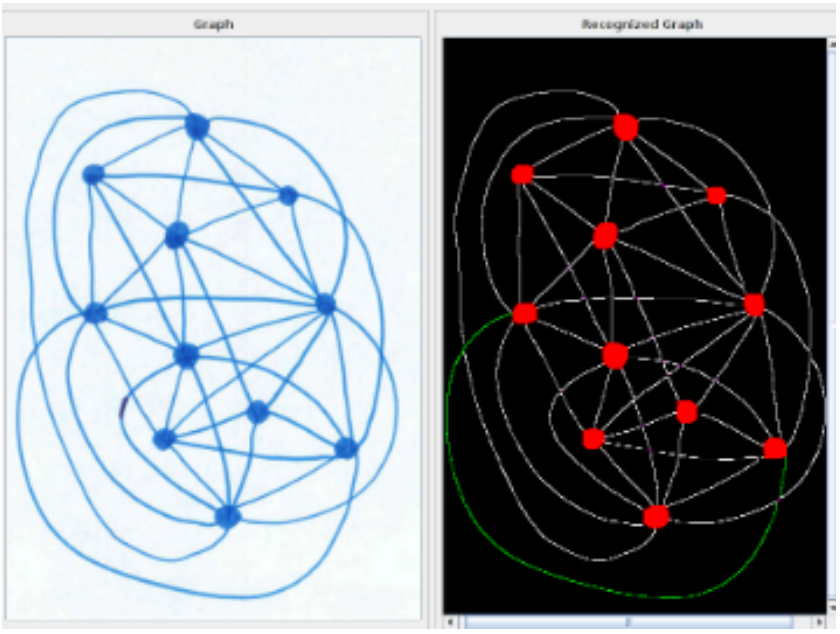


Figure 2.33: Graphical user interface of OGR [Auer et al., 2013]

2.5.4 Other Recognition Systems

Hammond and Davis [2005] designed a sketch description language that shows how sketch diagrams are edited, displayed and drawn. This language is called LADDER. After designing LADDER, its structural description was then transformed into domain-specific shape recognisers and shape exhibitors for use. This system can be used in numerous domains like UML diagrams and flowcharts. To build a sketch recognition system for a new domain using LADDER framework, the user must write a domain description which shows what the domain of interest symbols look like, and how they should be edited and displayed after they have been recognised.

In this framework, a shape definition is done in three phases, namely, the process of recognising the shape, how to display the recognised shape, and how to edit the recognised shape if need be. The translation phase parses the description and generates codes. These codes show how to recognise, edit and display different shapes once they are recognised. The user does not necessarily need to have a programming idea to perform sketch recognition. Once a user writes a description, the description is then automatically translated into shape recognisers and shape exhibitors. All these features are customisable to suit the domain they are being applied to. The display feature in this framework gives the user feedback about a symbol that was recognised and also gives a privilege for the beautification of the diagram. The language has an editing attribute and is made up of constraints, predefined shapes and display methods. LADDER requires its users to define the shape(s) they want to draw; this makes it cumbersome to use if the users do not have a good understanding of the domain and

also time-consuming to write or type for users that have an understanding of the domain. LADDER description contains over a hundred shapes. However, when choosing a shape, care must be taken. For example, if a user wants to draw an FA diagram and the user chooses an ellipse shape as a circle, this will not be recognised as a state in LADDER. Despite the efficiency of this system to recognise hundreds of shapes, it still fails to recognise shapes with lots of curves. All these drawbacks make LADDER not suitable for this research.

Ouyang and Davis [2009] proposed a new model for recognising sketch symbols. This system focuses on the visual appearance of an image. This allows the stroke level and visual varieties observed in freehand drawings to be better handled by the users. They also presented a symbol classifier that is effective and compatible with local deformations and rotation. Information about the temporal nature of strokes was investigated since the work focuses on online recognition approach. To eliminate the differences caused in strokes due to scale, sampling, and translation symbol normalisation was done on the images. Each stroke was resampled at a constant spatial frequency aiming at making the feature extraction process more reliable.

A major aspect of Ouyang and Davis [2009] work was on how the online stroke sequences were converted into a set of feature images with low resolution. This was done by looking at each sample point in the pen trajectory and computing five features for this point. One of these features focuses on the stroke endpoints, while the remaining four are concerned with stroke orientation. The outcome of this aspect is an ordered sequence of feature values, i.e. each point in the symbol has five features. In the next stage, the authors smooth and downsample the feature images. To do this, they used a Gaussian smoothing function on each image that spreads feature values to neighbouring pixels. This was done to ensure that small spatial variations in the symbol correspond to changes gradually in the feature values and also to improve the tolerance of the image to distortions and local shifts. During the symbol recognition stage, a deformable template matching algorithm was used. The proposed method correctly recognised 99.2% of the digits in the used corpus.

A major drawback in Ouyang and Davis [2009] approach and the used deformable template matching algorithm is that it requires a match between the input symbol against all of the training examples. Hence, matching the full set of image deformable model can take several seconds. In our work, we want to model a system that is as quick as possible. This makes this approach unsuitable for our work. A full sketch recognition system was not presented in Ouyang and Davis [2009]. In particular, image segmentation and extracting valid symbols from rough sketches and symbol detection were not addressed in the proposed method. In this research, we are interested in how

to fully recognise an FA diagram and understand the relationship that exists between the components of the FA diagram. Hence, image processing stages such as extraction and segmentation that are missing in Ouyang and Davis [2009] are key to our research. This makes their approach not applicable to this research.

Freeman and Plimmer [2007] discussed an approach for semantic and syntactic recognition of connectors (i.e. edges, arrows and lines). They used InkKit [Chung et al., 2005], an extensible tool for diagram recognition. It works with sketch recognition features used in most domains and allows an extension for domain-specific applications. The framework was used on undirected graphs, directed graphs, and organisation charts. For these three types of connected diagrams, Freeman and Plimmer [2007] identified the main semantic and syntactic requirements for effective recognition of connectors and identified similar functionality to be integrated into the final recognition system. The approach used here focussed on finding the components at the endpoints of a connector and tags each component with the connector information. The domain of interest module can trace the paths between components and uses path information as appropriate. The relationship between the image components used in Freeman and Plimmer [2007] was determined by finding overlaps because the edges of the components do not have a direction. Despite the simplistic approach used in the recognition of arrows in this work, the directionality of arrows was not addressed. Hence this approach will not work for an FA diagram as the direction of a transition is very important when designing and recognising an FA. Moreover, the idea used in this work identified the shaft of an arrow as the longest stroke and the arrowhead as a shorter stroke. Recognising arrows using this idea in offline systems where users have the freedom of drawing as they want before presenting it to the recognition system may be challenging. To have a better result, we did not use the approach presented in Freeman and Plimmer [2007]. Instead, we developed our arrow detector and recogniser, which is presented in Chapter 5.

Babalola [2015] proposed a system for automatic recognition and interpretation of machine-generated and printed FA diagrams. The system used most of the well-known image processing techniques for extracting information from the diagram pixels. They went further to use constraint multiset grammars [Marriott, 1994] to interpret the meaning of the diagrams. In their research, a fully working prototype was not implemented. Since the diagrams used in the research were acquired from textbooks using scanners of high quality, the images contain little or no ambiguities. This makes the problem easy as the data were carefully collected. Moreover, relationships that exist between the components of an FA diagram were not fully considered in their study. All the drawbacks highlighted above are key to developing a full recognition system. Hence,

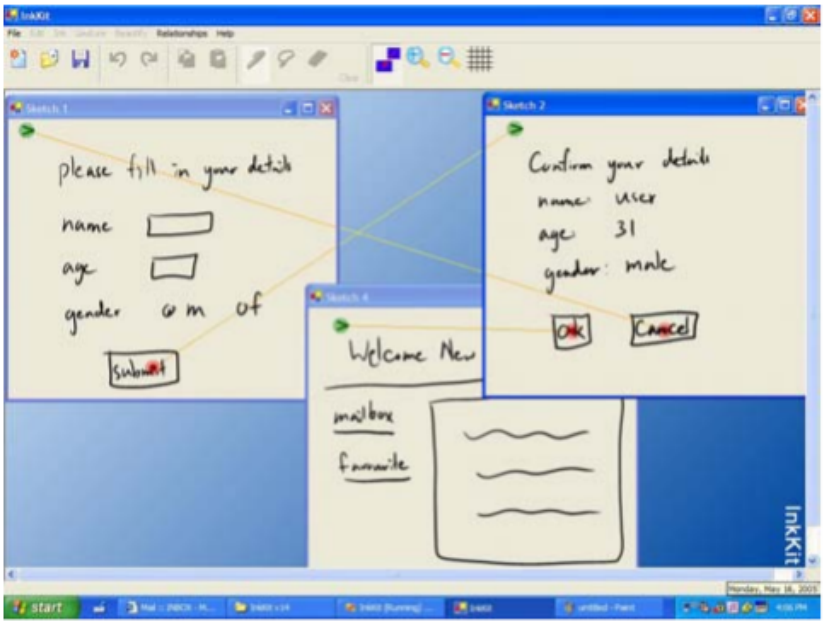


Figure 2.34: Sketches in InkKit Interface [Chung et al., 2005]

the approach proposed in their studies will not be sufficient for our research. This led to the introduction of our own recognition system.

Kara and Stahovich [2007] described an approach for the hierarchical parsing of hand-drawn sketch diagrams with focus on network diagrams. This work was based on a hierarchical architecture of different techniques. The first phase of the hierarchy aims at finding the arrows in the sketch. These arrows were referred to as markers because of two major attributes: First, their kinematic and geometric characteristics allow the extraction process to be easy and continuous from a group of strokes, and second, they serve as delimiters, which enables the remaining strokes to be carefully clustered into distinct groups corresponding to individual symbols. The main idea here is that stroke clustering is motivated by the identified arrows in the first phase, without the need for a search. Based on the outcome of the clustering algorithm, the next phase uses a contextual knowledge to generate a set of candidate interpretations for each of the stroke groups. The stroke groups are then evaluated using a symbol recogniser. This recogniser determines which of the interpretations is correct. The recogniser can learn new symbol definitions from a single prototype example provided. This is the main advantage of their recogniser.

In summary, a group of strokes was first examined and identified as arrows in Kara and Stahovich [2007]. The identified arrows then employ spatial analysis to group the uninterpreted strokes into different clusters, each representing a single object. The spatial analysis is then used to inform a trainable shape recogniser on how to best interpret the different clusters. Based on the idea developed in Kara and Stahovich

[2007], SimuSketch was designed. SimuSketch is a sketch-based interface for Matlab's Simulink software package. The result of the experiment performed on the sketch interface shows that novice users can easily use the system to solve engineering problems without having to know much about the recognition approaches. The focus of the approach proposed here is to recognise arrows drawn as a single stroke, i.e. arrow shaft and head are drawn at a single pen movement. This makes it not suitable for our research as our findings showed that most users would draw an FA arrow or transition as two separate strokes, i.e. the shaft and the arrowhead. Hence, it is important to have a system that can identify and recognise arrows drawn as a single stroke and those drawn as two strokes.

Some other work in the literature described shape descriptors [Kim and Kim, 2000; Roy et al., 2018]. Most of these works proposed that having a good shape descriptor is a key part of multimedia content description since shape is a basic property of an object. Zhang and Lu [2002] proposed a region-based shape descriptor. In this type of descriptor, all the pixels in a shape region are considered when obtaining the shape representation. Geometric Moment Descriptors (GMDs) were described in Zhang and Lu [2002]. The GMD approach is based on geometric moment invariants for shape representations. The moment of shapes are used to get the moment invariants and they are invariant to 2D geometric transformations of shapes. In this descriptor, a feature vector is made up of seven moment invariants: $f = (\Phi_1, \Phi_2, ..., \Phi_7)$. These seven normalised geometric moments are invariant under rotation, translation and scaling. This descriptor was used in this research because it is computationally effective and compact to shape representation. They are presented in Zhang and Lu [2002] as follows:

$\Phi_1 = \gamma_{20} + \gamma_{02}$

$\Phi_2 = (\gamma_{20} - \gamma_{02})^2 + 4(\gamma_{11})^2$

$\Phi_3 = (\gamma_{30} - 3\gamma_{12})^2 + (3\gamma_{21} - \gamma_{03})^2$

$\Phi_4 = (\gamma_{30} + \gamma_{12})^2 + (\gamma_{21} - \gamma_{03})^2$

$\Phi_5 = (\gamma_{30} - 3\gamma_{12})(\gamma_{30} + \gamma_{12})[(\gamma_{30} + \gamma_{12})^2 - 3(\gamma_{21} + \gamma_{03})^2] + (3\gamma_{21} - \gamma_{03})(\gamma_{21} + \gamma_{03})[3(\gamma_{30} + \gamma_{12})^2 - (\gamma_{21} + \gamma_{03})^2]$

$\Phi_6 = (\gamma_{20} - \gamma_{02})[(\gamma_{30} + \gamma_{12})^2 - (\gamma_{21} + \gamma_{03})^2] + 4\gamma_{11}(\gamma_{30} + \gamma_{12})(\gamma_{21} + \gamma_{03})$

$\Phi_7 = (3\gamma_{21} - \gamma_{30})(\gamma_{30} + \gamma_{12})[(\gamma_{30} + \gamma_{12})^2 - 3(\gamma_{21} + \gamma_{03})^2] + (3\gamma_{12} - \gamma_{03})(\gamma_{21} + \gamma_{03})[3(\gamma_{30} + \gamma_{12})^2 - (\gamma_{21} + \gamma_{03})^2]$

In this section, we have described some existing handwriting recognition systems. Most of the systems explained here are based on online recognition approaches. This can be due to the complexity involved when developing an offline recognition system as explained in Section 2.1 and 2.6.1.2. We have also explained what has not been done in the field of handwriting recognition by highlighting the drawbacks of each of

the systems explained above. The highlighted drawbacks are the reason why all the explained systems are not suitable for our research.

2.6 Handwriting Recognition

Handwriting is a skill that is personal to individuals. According to Plamondon and Srihari [2000], the underlying characteristics of handwriting can be classified into three groups –

- an artificial graphical mark,
- the purpose of the mark, i.e. to pass information, and
- the achievement of the purpose, i.e. by virtue of the mark's conventional relation to language.

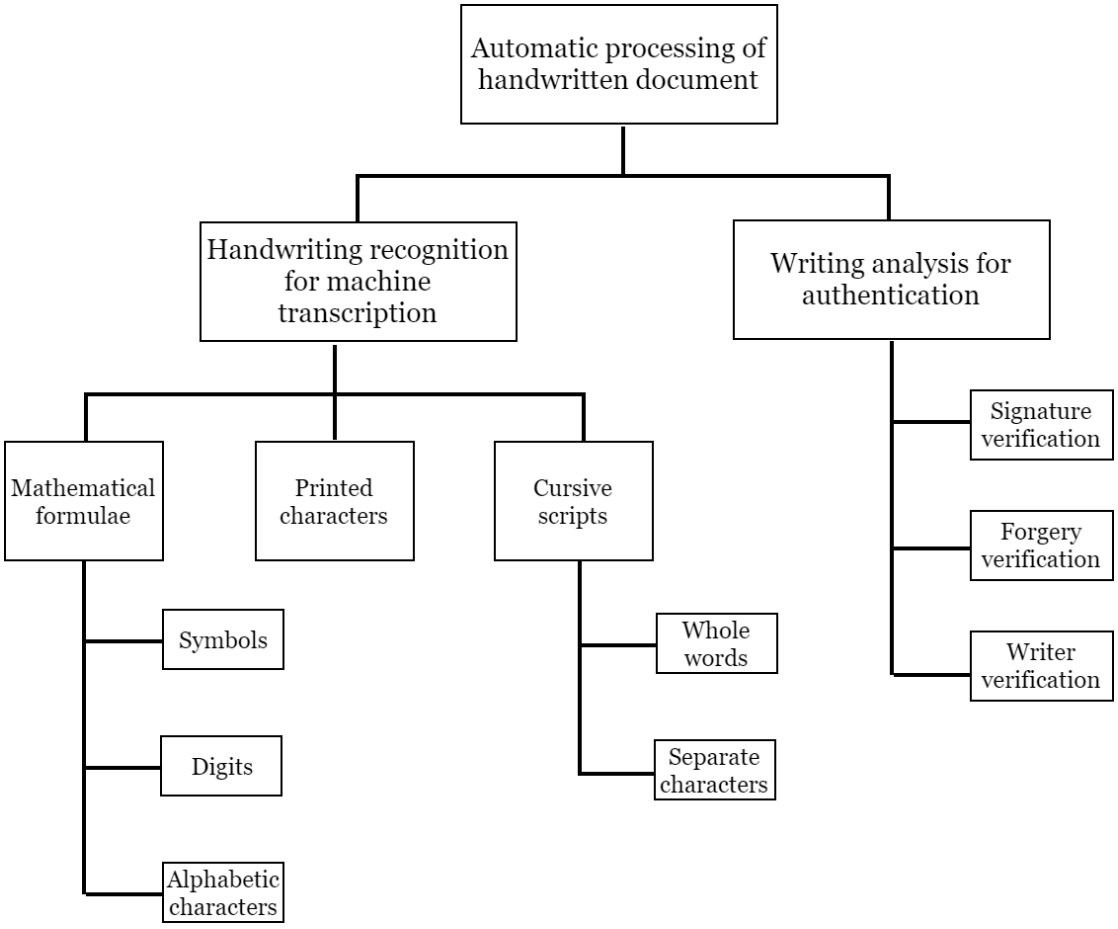


Figure 2.35: Handwriting processing categories

As illustrated in Figure 2.35, the processing of handwriting can be divided into two, namely, handwriting recognition and handwriting analysis. Handwriting recognition,

which is the focus of this research, focuses on producing output for machine transcription. Handwriting analysis focuses on using handwriting for authentication purposes. In this research, our aim is on handwriting recognition. Numerous kinds of analysis, recognition and interpretation can be connected to handwriting. The process of converting a language represented in its spatial form of graphical marks into its symbolic representation is referred to as handwriting recognition. The process of determining the content of handwriting is known as handwriting interpretation [Plamondon and Srihari, 2000]. Assuming that individual handwriting is unique, the task of identifying the author or writer of handwritten texts among a group of writers is referred to as handwriting identification. The main goal of handwriting recognition is to output the best interpretation of a given sequence of handwritten strokes. Despite the work that has been done in handwriting recognition, there are still some challenges in recognition. The subsequent sections in this chapter highlight some of these challenges.

2.6.1 Types of Handwriting Recognition

The task of recognising handwritten documents can be done in two ways as explained earlier. This task is based on the mode of input data, namely, online recognition and offline recognition [Zhang et al., 2017]. In online recognition, the trajectories of pen ball movements are noted and analysed to understand and identify the information being expressed. This approach has been widely used for pen input devices, personal digital assistants, smartphones and computer-aided education. In offline recognition, handwritten documents are analysed and classified into different categories or classes. The offline approach has been used in numerous applications, such as mail sorting, bank cheque reading, and book and handwritten notes transcription. As explained in Chapter 1, the focus of this research is on using offline approaches to recognise hand-drawn FA diagrams.

2.6.1.1 Online Recognition

Online recognition refers to the methods and approaches that deal with the automatic analysis of handwritten documents as they are written in real-time using a screen (tablet) and a stylus. This recognition approach shows the information about the pen-ball such as its velocity, position or pace as a function of time. Compared to the offline approach, online recognition of handwriting has gained a lot of popularity in the last few decades and has been widely used [Tappert et al., 1990; Samanta et al., 2018; Ren and Ganapathy, 2019]. The invention of smartphones has also contributed to this popularity.

During the recognition process in an online approach, the input data is obtained from pen-based devices such as tablets, smartphones, and touch screen PCs. It may be required to pre-process these data to reduce unwanted noise (see Section 2.4.2) that might have originated from the data sources such as the capturing device. The pre-processed data can then be segmented into smaller components, e.g. letters or strokes. Sometimes, the segmentation process may be optional. The inclusion of the segmentation stage during recognition is based on the adopted recognition approach, as it is a very important stage when using an analytic recognition approach whereas it can be skipped if a holistic recognition approach is being used [Al-Helali and Mahmoud, 2017]. The feature extraction phase aims to represent input data with a set of attributes called features. These features must be discriminative to distinguish the data of different categories or classes during the classification stage. The data to be used during the recognition stage can be split into training, validation and test sets. However, in some cases, the validation dataset can be ignored because it is part of the training set. The training dataset is used to train and build the classifier. Sometimes, post-processing can be done to improve the recognition result.

Notwithstanding the success rate of online recognition, there are still some unsolved problems in this recognition approach [Keysers et al., 2016], such as;

- there exists a strong variability in writing even if the document is written by the same writer. Factors like sloppiness, input speed, and using a mobile phone while walking can cause this,
- there exists a strong variability in writing style between different individuals,
- various ambiguities can be acquired during data acquisition. For example, a small stroke indicating a comma could be mistaken for noise, and
- for scripts with various characters, training a machine to recognise this is still challenging because the gathering of a proper training dataset may be cumbersome.

2.6.1.2 Offline Recognition

Offline handwriting recognition is the process of converting an image containing handwritten text or diagrams into a machine-readable version of that text or diagram. The main and most common tasks performed using the offline approach are character recognition [Zhong et al., 2015; Elleuch et al., 2016] and word recognition [Poznanski

and Wolf, 2016; Roy et al., 2016]. Due to the computational complexity of offline recognition approaches, research in the field is still lagging compared to online handwriting recognition.

In these approaches, image processing components such as image acquisition, preprocessing, segmentation, classification and recognition are used. All of these components depend on one another to achieve a good accuracy and recognition result. Most of these components have been explained in Section 2.4.

As discussed earlier, some research studies have been done in the field of offline handwriting recognition. Zamora-Martinez et al. [2014] contributed to the recognition of unconstrained offline continuous handwritten text which is generally seen as a difficult task. They used neural network language models in decoding based on hybrid hidden Markov models (HHMM) and bidirectional recurrent neural networks (BLSTM). These two models, i.e. HHMM and BLSTM, were then combined in their experiments. Experiments with the algorithms were performed on the IAM offline database. The outcome of the experiments performed on the IAM database indicated that a consistent reduction in word recognition error rate can be obtained using neural network language models when compared with statistical N-gram language models in Poznanski and Wolf [2016]. Rashid et al. [2011] proposed an approach using HMM approaches to evaluate OCR systems with low-resolution text. The text used is classified into two different categories, namely, screen rendered text-lines and screen rendered isolated characters. A text-line was described as a sequence of text characters and each character was represented by a sequence of observations O, i.e. $O = o_1, o_2, ...o_N$. Du et al. [2016] introduced a segmentation free technique for offline handwritten Chinese text recognition. The technique uses deep neural networks (DNN) based on HMM. During the feature extraction, the gradient-based features were extracted using the DNN classifier for each of the text frames or windows. HMM was then used to model the text line sequentially. The major reason for using the DNN with HMM in Du et al. [2016] was to compute the posterior probability of all HMM states.

2.7 Handwriting Databases

Handwriting databases are databases of handwritten text characters or hand-drawn images. There exist some databases in literature for handwriting. However, most of these databases are based on the online recognition approach and few on offline recognition. The segmentation of datasets acquired using an offline recognition approach is a difficult and cumbersome task. This may be one of the reasons there are only a few offline databases.

2.7.1 IAM Database

The IAM database has been widely used for training and testing of handwritten text recognisers. It can also be used for identification and verification of handwriting experiments.

2.7.1.1 IAM Handwriting Database

The IAM handwriting database [Marti and Bunke, 2002] is a database for English sentences developed for offline handwriting recognition. The database is based on the Lancaster-Oslo/Bergen (LOB) corpus. The database contains 82,227 examples of handwritten words spread across 9,285 lines of text taken from approximately 400 writers. All the texts in this database are unconstrained handwritten texts (i.e. the writers were not given any instructions before writing). Each document was scanned at 300dpi resolution and saved as a PNG file with 256 gray levels. The words have been extracted from the backgrounds (pages) of scanned text using an automatic segmentation technique and they were all verified manually.

2.7.1.2 IAM On-Line Handwriting Database

The IAM on-line handwriting database (IAM-OnDB) [Liwicki and Bunke, 2005] consists of handwritten English text acquired on a whiteboard. The database contains 86,272 examples of words collected from 11,059 dictionaries written by 221 writers. All the contents of the database are unconstrained handwritten text acquired with the E-Beam system. All the collected data were stored as XML-files.

2.7.2 MNIST Database

The MNIST database [LeCun et al., 1998] of handwritten digits has a total of 60,000 examples for the training set and 10,000 examples for the test set. This database is generally referred to as a standard database for researchers trying to learn pattern recognition or learning techniques and their applications on real-world data because little effort is required for pre-processing and formatting.

2.7.3 Flowcharts

Most of the hand-drawn flowchart recognition works have used the online database created by Awal et al. [2011] for evaluating their algorithms. Recently, Bresler et al.

[2014] developed another database for flowcharts. Awal et al. [2011] proposed an online handwritten flowchart database. The database contains 419 diagrams of flowcharts annotated at stroke and symbol levels. The problem of online hand-drawn flowchart recognition was addressed using two different approaches:

(i) all diagrams were considered to be segmented correctly and two different classifiers were proposed to perform the text and non-text separation and the graphical component recognition. The text and non-text classifier achieved an accuracy of 90% and 98% respectively and about 93.5% in the graphical component recognition, and

(ii) a global approach was proposed for the flowchart segmentation and recognition task.

Bresler et al. [2014] created an online handwritten flowchart database. The database contains 672 samples of flowchart diagrams with 28 diagram patterns (i.e. 28 different flowchart structures were drawn) drawn by 24 writers. The diagrams were split into training, validation and test datasets. All the diagrams were annotated and the relationship among the components was known, unlike the case of Awal et al. [2011] where there was no data annotation and only individual symbols were identified.

2.8 Techniques used in Handwriting Recognition

As explained in previous sections, handwriting recognition is a very important task in the image processing field with ongoing research. In this section, we will explain some techniques and approaches that have been used in the recognition of handwritten documents and images. The scope of this section is within the domain of handwriting recognition and not other image recognition tasks even though most of the techniques explained here can be applied to different types of images, speech and other types of object recognition [Ma and Guo, 2014; He et al., 2017].

2.8.1 Support Vector Machines

SVMs [Cortes and Vapnik, 1995; Chang and Lin, 2011] are one of the most used supervised learning algorithms for regression, classification and other learning tasks. The basic idea of this classifier is to construct a hyperplane as the decision plane. This plane separates the positive and negative patterns with the largest margin. The aim of

the hyperplane is to group new data examples into different classes. An SVM hyperplane is a line separating a plane into two classes where a class is on one side and the other on another in a two-dimensional space. The main idea of SVMs is to construct a non-linear kernel function to map the data from an input pattern space into a possibly high-dimensional feature space and then generalise the optimal hyperplane with the highest margin between the patterns of the two categories. Hence, it is used for binary (positive or negative) classification.

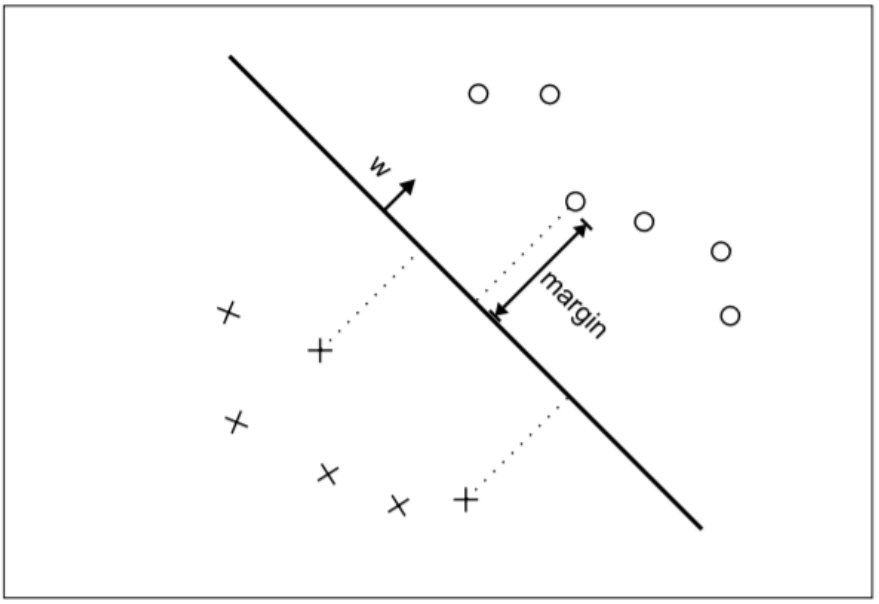


Figure 2.36: A simple linear Support Vector Machine [Tong and Koller, 2001]

According to Cortes and Vapnik [1995], the method for optimal hyperplanes for separating the training dataset is as explained below.

Optimal Hyperplane Algorithm The set of labeled training patterns

$$\{(p_1, q_1), ..., (p_l, q_l)\}, \quad p_i \in \{-1, 1\}, \qquad \text{(Equation 2.20)}$$

is said to be separable linearly if a vector x and a scalar b exist such that Equation 2.21 is true for all the training sets in Equation 2.20:

$$\begin{aligned} x.q_i + b &\geq 1 \qquad \text{if } p_i = 1, \\ x.q_i + b &\leq -1 \qquad \text{if } p_i = -1. \end{aligned} \qquad \text{(Equation 2.21)}$$

The inequalities in Equation 2.21 can be expressed further as:

$$p_i(x.q_i + b) \geq 1, \qquad i = 1, ..., l, \qquad \text{(Equation 2.22)}$$

The optimal hyperplane is

$$x_0.q + b_0 = 0 \qquad \text{(Equation 2.23)}$$

The optimal hyperplane separates the training sets with a maximal margin. This determines the direction $x/|x|$ where a maximal distance exists between the training vectors of two different classes Cortes and Vapnik [1995].

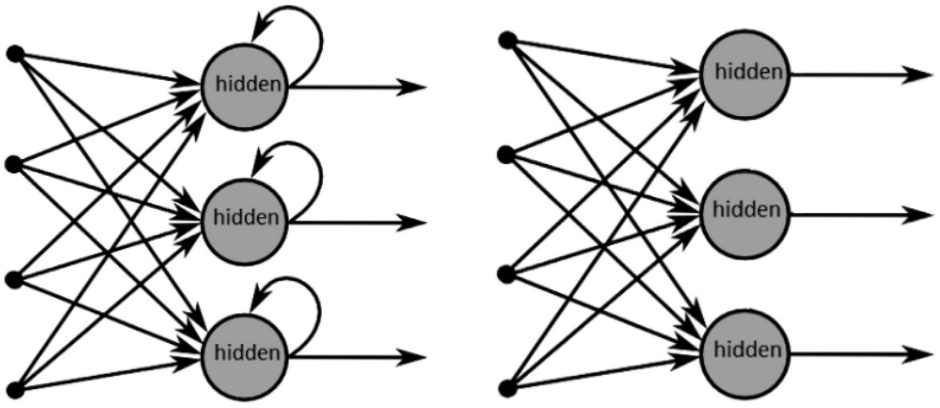
SVMs have been used in numerous handwriting recognition research studies. Nasien et al. [2010] proposed an approach for recognising handwritten English characters (lowercase and uppercase) using the Freeman chain code (FCC). FCC was used as the representation method of an image character. During the character classification, SVM classifiers based on statistical learning theory were used. All the SVMs were trained with data from the NIST database. The training feature sets and the results were explored by using separate test data which are lowercase and uppercase characters. The SVM classifier proposed in the work was trained with hand-written characters from the NIST database. This makes it not applicable to our research.

Ebrahimzadeh and Jampour [2014] proposed an appearance feature-based technique using Histogram of Oriented Gradients (HOG) [Dalal and Triggs, 2005] and SVMs for handwritten digit recognition. During their experiments, HOG was used as a feature descriptor for the digits. They employed SVMs as classifiers. The proposed method was analysed and tested on the MNIST dataset and it achieved an accuracy of 97.25%. Simistira et al. [2015] trained a probabilistic SVM classifier for recognising online handwritten mathematical formulas. The focus of their work is to recognise the spatial relationships between two mathematical expressions or symbols and then employ the Cocke-Younger-Kasami (CYK) algorithm [Firsov and Uustalu, 2014] (a parsing algorithm for context-free grammars) to parse the symbols or expressions in order to produce the respective mathematical markup language (MathML) output. The approach described here was evaluated on the MathBrush database. The proposed technique does not consider the relationships that exist between characters. This criterion is important in recognising FA diagrams, hence, we cannot adopt the approach in our study.

2.8.2 Recurrent Neural Networks

RNNs are feedforward neural networks (FNNs) augmented by the inclusion of edges that span adjacent time steps and introduce a notion of time to the model [Lipton et al., 2015]. Unlike FNNs, RNNs may not have cycles among their conventional edges. Edges that join adjacent time steps (known as recurrent edges) can form cycles. An RNN consists of input units $\{x_0, x_1, ..., x_t, x_{t+1}\}$, output units $\{y_0, y_1, ..., y_t, y_{t+1}\}$ and

the hidden units $\{s_0, s_1, ..., s_t, s_{t+1}\}$ as shown in Figure 2.38 [Zhang et al., 2018]. Figure 2.37(a) illustrates that the hidden neurons in an RNN can receive input values from both the hidden layer and the input layer, while Figure 2.37(b) illustrates that an FNN can receive input values only from the input layer.



(a) Recurrent neural network (b) Forward neural network

Figure 2.37: RNN versus FNN [De Mulder et al., 2015]

Figure 2.38 illustrates an RNN. At time step t, the RNN takes the current sample x_t and the previous hidden representation s_{t-1} as input to get the current hidden representation s_t:

$$s_t = f(x_t, s_{t-1}), \qquad \text{(Equation 2.24)}$$

where f represents the encoder function.

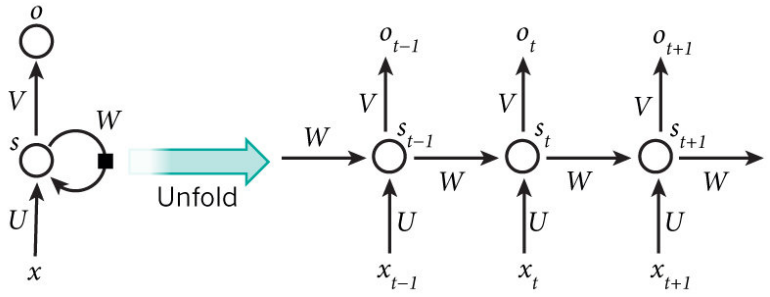


Figure 2.38: Recurrent Neural Network [Zhang et al., 2018]

2.8.2.1 Bidirectional Long Short-Term Memory

A Bidirectional Long Short-Term Memory Recurrent Neural Network (BLSTM-RNN) [Schuster and Paliwal, 1997] processes the sequence in both directions as illustrated in Figure 2.39. There are two recurrent layers in this network: a forward layer (takes its input from the previous timestep), and a backward layer (joined or connected to the

next timestep). As stated in the previous section, LSTMs have the ability of long time lag, resolve temporal distances. However, all the states in an LSTM are accumulated forward. When dealing with the handwriting recognition problem, it is important to consider backward information, hence the need for BLSTMs. For example, a cursive handwriting script will always have all its characters connected to one another. Moreover, to identify a given text character, it is important to know the letters on both sides (right and left) of the text. Thus, this makes the previous (backward) and next (forward) characters important.

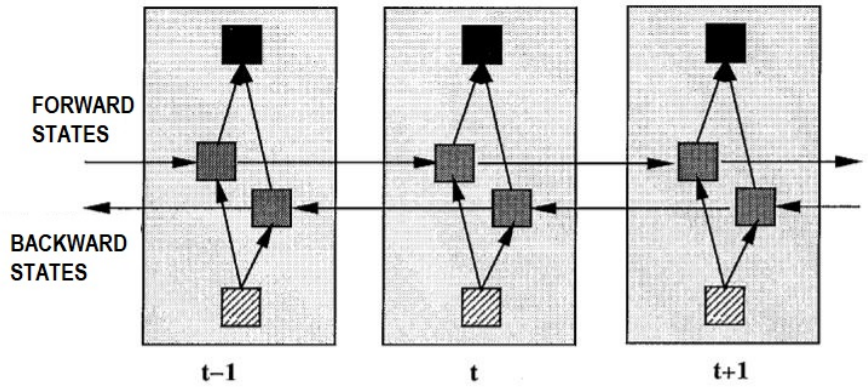


Figure 2.39: General structure of the bidirectional recurrent neural network (BRNN) shown unfolded in time for three time steps [Schuster and Paliwal, 1997]

Chen et al. [2015a] proposed an approach that uses a context-sensitive-chunk (CSC) based back-propagation through time techniques to train an LSTM-RNN for offline handwriting recognition. LSTM was used to split each training sequence into chunks with appended contextual observations for character modelling of offline handwriting. The proposed handwriting recognition system is based on an NN-HMM hybrid system. A forced alignment given by a Gaussian mixture model HMM system was used to provide the frame-level state targets. Softmax was used as the activation function of the BLSTM's output layer. The unit number of the softmax is the total number of HMM states. During training, a frame-level cross-entropy objective function is minimised. A constant learning rate is employed alongside a validation set to monitor the training process. CSCs of a mini-batch can be chosen from different parts of different sequences randomly in the approach proposed in Chen et al. [2015a]. However, it was noticed during their experiments that there is no much difference even without mini-batch. During recognition, BLSTMs are hybrid with HMM state transition probabilities and language model scores to know the result of the recognition process. The algorithm used in their work was evaluated on the IAM offline handwriting database. Also, the experiments were conducted on deep LSTM and deep BLSTM.

Frinken et al. [2014] developed a self-training BLSTM-RNN to find keywords in handwriting. The work presented here shows that if a handwriting recogniser trained

with labeled data (cumbersome and tedious approach of gathering data) exist then a self-training approach can be adopted to recognise unlabelled data (data that can be gathered easily at little or no cost) by adding words that have been recognised with high confidence to the training dataset for re-training. The approach was evaluated on historical handwriting. The result of the experiment showed that labeling of data is not always compulsory. Messina and Louradour [2015] used multi-dimensional LSTM-RNNs for recognising lines in Chinese handwritten text. The segmentation approach used here is different from most of the approaches used in Chinese character recognition. According to Messina and Louradour [2015], most of the prior works performed pre-segmentation of Chinese text images into characters, which can be seen as a drawback. The LSTM proposed here is used for modeling sequences of characters. The input to the system is a diagram of a text line that is scanned in four directions, producing predictions for an individual character in the charset. The scanning was done to divide the input into regularly spaced frames. To each of the frames in the diagram, an output vector of probabilities is produced. The network free parameters are trained to minimise the negative log-likelihood of the sequences of characters predicted by the network and the target transcriptions for all diagrams in the training dataset. During the experiment, the raw outputs of the network allow a character error rate of 16.5% and the use of a language model at the character level improved the performance to 10.6%.

Bluche [2015] did a study of different areas of optical models using DNNs in the hybrid NN scheme. The study showed that DNNs can output significant and consistent results over networks with one or two hidden layers. We used BLSTM-RNN during the text and graphics separation to capture the global context of the text to text and text to graphics classification. The probabilistic outputs from the two BLSTM neural networks are combined to obtain the final labeling probability reported in Chapter 5.

2.9 Summary

A review of existing systems, algorithms and approaches with a focus on handwriting recognition has been given in this chapter. Since research in offline handwriting (text and images) recognition is still trying to gain recognition when compared to online recognition, most of the approaches, algorithms and recognition systems described in this chapter are based on online recognition approaches. In this chapter, we have discussed image pre-processing, segmentation, classification and some existing handwritten images databases. The purpose of this research is to recognise hand-drawn FA

diagrams. So, most of the approaches discussed here have been used for images such as flowcharts, and UML diagrams.

Image pre-processing is a crucial phase in offline recognition. In this thesis, noise removal, image enhancement, thresholding and thinning are the pre-processing approaches used. When developing a recognition system, the segmentation phase has been generally known to be one of the most cumbersome and challenging phase. To tackle this, most work in handwriting recognition focus on limiting their segmentation algorithm to their domain of interest. By doing so, they have limited components to segment, hence, improving the result of their algorithm. In this research, we focus our segmentation algorithm on FA diagrams only. BLSTM-RNN will be used in this thesis for separating the graphical components of an FA diagram from the textual components. SVMs will be used to classify the different states of an FA diagram to the classes each of them belong to. In the next chapter, we present an overview of the research methods used in this thesis.

Chapter 3

Methodology

3.1 Introduction

The previous chapter highlights and explains the background and related work to this study. It shows the various image processing techniques and algorithms relevant to this study. The chapter also described the limitations of some existing recognition systems. A major identified drawback in the literature is the lack of datasets for offline recognition systems. Most of the datasets found and reported in the literature are acquired using online approaches. Another drawback is the segmentation and recognition of handwriting or hand-drawn images acquired using offline approaches. These images do not contain temporal information which makes it difficult to obtain information about the type of input. In contrast to offline systems, temporal information like the speed of the writing, the direction of each stroke, order of the strokes and the number of strokes are captured in online systems. This is one of the reasons such systems are considered to be more accurate than the offline system. This temporal information can be useful in differentiating between similar shapes, like when comparing *S* to *5*. If the shape is drawn with two strokes, it is probably a *5*. The process of knowing the number of times a shape strokes are drawn can easily be known by the temporal information of slowing down or pausing at the corners. This process is however not known during offline recognition. So a stroke drawn as *5* might be interpreted as *S* in offline systems.

Segmenting handwritten images obtained offline remains a significant and open problem in the field of handwriting recognition [Graves and Schmidhuber, 2009; Wu et al., 2015]. This is because the components of the input diagram can consist of one or more strokes, and it is often not clear when deciding which strokes should be grouped

together. Examples of segmentation ambiguities are image components may be re-traced when drawing and the algorithm assumes it is noise. The character-within-character problem is also another common occurrence, i.e. a handwritten lowercase character *d* might be interpreted as *cl* if drawn with two strokes that are somewhat isolated from each other. All these identified problems are addressed in later chapters of this thesis. These problems have been put together and we came up with a new approach to recognise hand-drawn diagrams. In this research, we focus on hand-drawn FA diagrams.

Recognition of hand-drawn images has been one of the most important aspects of the field of digital image processing. Recently, hand-drawn image recognition systems are being applied in many domains [Chen et al., 2015b; Di Martino and Esposito, 2016; Carbune et al., 2019]. This research focuses on recognising FA diagrams acquired using offline recognition approaches. The main aim of this work has been stated in Section 1.3 as to present an offline recognition approach to recognise hand-drawn FA diagrams and generate their corresponding transition tables. The choice of some of the techniques we used (pre-processing, segmentation, thinning and arrow detection) is an important task which must satisfy both effective pre-processing and thinning while preserving the important features of the input image and also be effective during the recognition process.

This chapter gives a detailed methodology of this research work. The architecture of the system is shown in Figure 3.1. Each phase in the architecture of the recognition system is explained in detail in Chapter 4 and Chapter 5. In this work, we present a multi-phase recognition system for hand-drawn FA diagrams. For ease of reading, this chapter begins by introducing the dataset and then describing our proposed methodology to achieve the research aim. Some of these proposed methods are modified from existing algorithms. Also, each of the phases in the architecture is expected to take an input and output a more useful version of its input for further processing. Throughout this thesis, arrows and transitions, circles and normal states mean the same thing.

3.2 Data Acquisition

Having adequate training and test datasets is a key requirement in developing a good recognition system, aside from using the right classification algorithm. It also assures that a good number of data are available to learn the decision boundary as a functional mapping between the feature vectors and the correct class labels. In this work, two datasets were created and used, namely, synthetic and raw data. These data are called synthetic (DB_a) and raw (DB_b) because of their mode of collection. The synthetic data

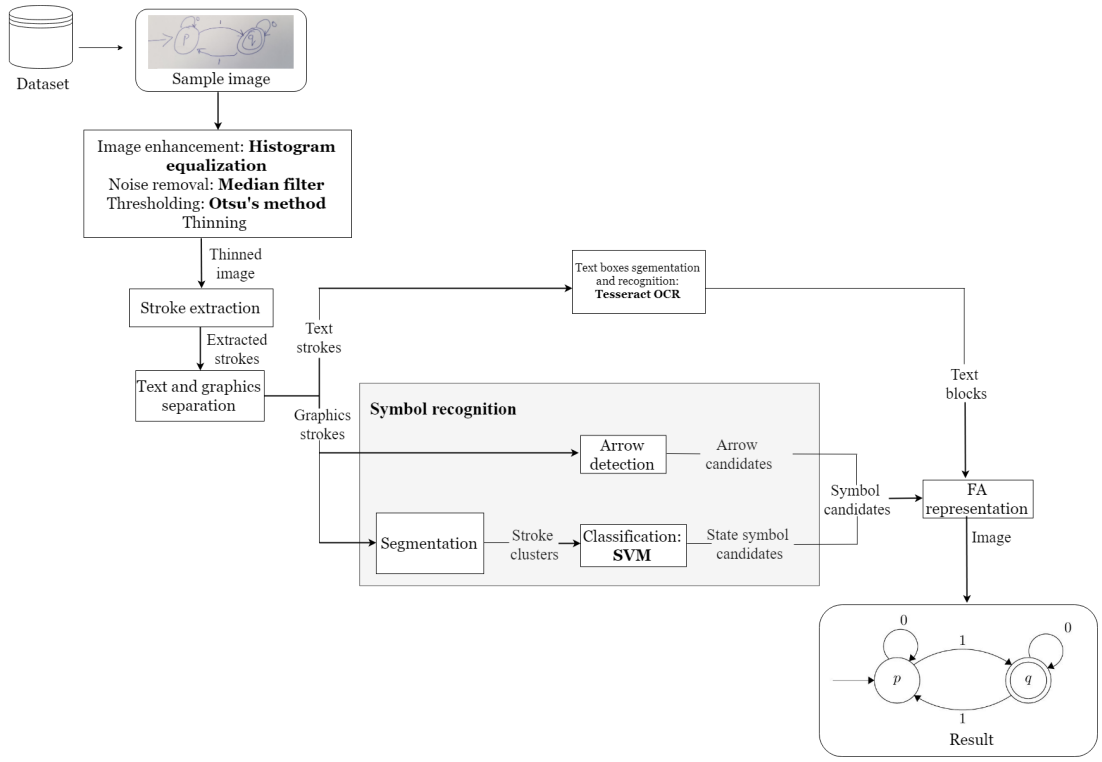


Figure 3.1: Overview of the recognition system

contains little or no irregularities due to their mode of collection which is explained later in this section. The raw data was acquired and used in this work to investigate how fast the recognition system can learn to process and recognise less carefully drawn FAs. Also, at the time of writing this thesis, there was no existing annotated dataset for offline FA diagrams to the best of our knowledge. We have collected and annotated our own data. These data were used to train and test the accuracy of the implemented system. The purpose of annotating the data is to be able to get information about the relationships between the components of an FA diagram, to know what each component represents and looks like.

3.2.1 Synthetic Data

These data are named synthetic because they contain little or no error due to their mode of collection. These diagrams were drawn by 50 students who have taken and passed a course in Formal Language and Automata Theory. To ensure that they all draw similar diagrams, ten different FA diagrams were drawn on a whiteboard and they were asked to reproduce all the diagrams on the provided plain white A4 paper. All students were instructed to draw naturally and were asked to sign a consent letter before drawing (see Appendix B.1 for consent letter). At the end of the collection, we had a total of 500 hand-drawn FA diagrams. The data was split into 350 training images and 150 test

images. Some of these ten different FA diagrams are presented in Appendix A.1 exactly as they were drawn by participants.

3.2.2 Raw Data

These data are called raw because they are collected from student past examination papers. This was done to investigate how the implemented system can adapt and learn to recognise a less carefully drawn FA diagram. Most of the data collected here contain drawing irregularities such as the components of the diagram not connecting properly (i.e. placing an arrow far away from a state), text labels intersecting graphics components, strokes overlapping, drawing a single stroke of a component several times hence making it difficult to find the pen trajectory of this stroke. Also, in this data, there was no consistency in the layout of the diagrams and there were many variations in writing direction, e.g., a user might start drawing a start arrow from the left (see Figure A.5(c)) while another starts drawing the same arrow from the top (see Figure A.5(a)). There was also a huge variation in drawing styles. All these challenges significantly made the recognition problem complicated. This data was collected from 100 different examination papers written in the same academic year. We chose two different drawings of FA diagrams from each paper. We have 200 hand-drawn FA diagrams. Ethical clearance was obtained before embarking on this task (see Appendix B.2 for research ethics certificate). Some examples of these two different FA diagrams are presented in Appendix A.2 exactly as they were taken from the examination paper.

During data acquisition, we noticed that people draw diagrams quite differently. Some like to draw from top to bottom and others like to draw from left to right. Also, some students prefer to finish drawing all the components first (start and final states) before adding the arrows and sometimes text or labels. However, most participants drew component by component and a few of them drew interspersed symbols i.e. scattering or drawing here and there of symbols. Interspersed symbols were mostly found in the raw data. Table 3.1 shows a summary of these datasets.

Table 3.1: FA datasets summary

Dataset	Writers	Patterns	Diagrams
DB_a	50	10	500
DB_b	100	2	200

A detailed analysis of the results of the experiments performed on the datasets is presented in Chapter 6.

3.3 Pre-processing

Pre-processing steps are the first process during image processing as shown in Figure 3.1. These steps are applied to reduce noise in the input data. The input to the system is hand-drawn FA diagrams (see Figure 3.2). These diagrams are obtained from different students. They are captured using a mobile phone camera which makes these diagrams prone to noise, distortion, and bad illumination. These diagrams are stored in our database of hand-drawn FA diagrams. During pre-processing, the amount of

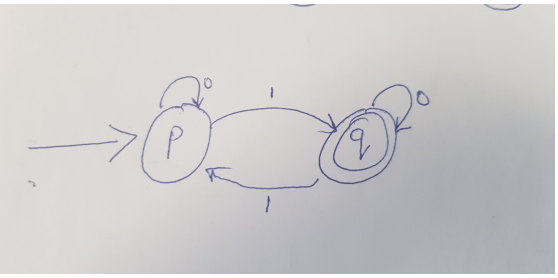


Figure 3.2: A sample FA input RGB diagram from DB_a

effort that is needed to be invested in this process is based on the available data. If the data have been gotten from a device that does not produce any noise, then there is nothing to do in this step. In our case, these diagrams were obtained from different students. They are photographed using a mobile phone camera which makes these diagrams prone to several irregularities such as noise, distortion, and bad illumination. This makes pre-processing an important step in this work. As stated earlier, these diagrams contained irregularities which need to be removed to improve the quality of the input data. The improved image is then thinned.

To improve the quality of the input image, we performed three operations. Firstly, image enhancement was done by adjusting the brightness and contrast of the input images automatically using their histograms (i.e. histogram equalisation). This is presented in Section 4.2. Secondly, we used a filter of 3×3 window and a standard deviation of $\sigma = 3.0$ to remove most of the noise from the input FA images. This is presented in Section 4.3. A median filter was used because it is much more efficient than the averaging or mean filter at removing any type of noise, especially salt-and-pepper noise. Hence, it was suitable for this research. Thirdly, Otsu's method was then used to perform automatic image thresholding with an initial estimate threshold value of 155 (see Section 4.4). We then performed image thinning to simplify the lines that make up the FA diagrams into lines of one pixel wide. We adapted the thinning approach presented in Boudaoud et al. [2015]. In this approach, one pixel (i.e. 7) was added to the edges of the input image as opposed to 6 that was used in Zhang and Suen [1984] (see

Section 2.4.6.1 for Zhang-Suen algorithm). This is to make sure that more edges are considered during thinning (see Section 4.5).

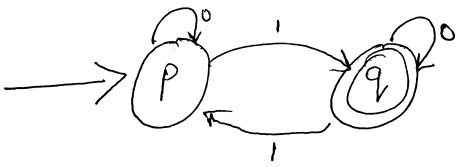


Figure 3.3: Result of performing pre-processing on Figure 3.2

3.4 Text and Graphics Separation

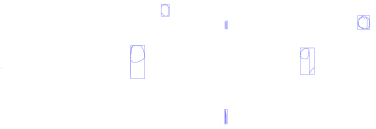
As illustrated in Figure 3.1, after the pre-processing steps, the system outputs a thinned image. The output will then be used as input during the text and graphics separation phase. During this phase, all the text strokes are taken out of the whole diagram, and the whole diagram is recognised without the text. Strokes here are grouped into two categories: graphics or text. Strokes that are classified as text are in blue in Figure 3.4(a).



(a) Text strokes are classified in blue



(b) Graphical components after removing the text strokes



(c) Textual component after removing the graphics strokes

Figure 3.4: Separation of the text and graphical components of Figure 3.3

In Figure 3.4(b) the text strokes are removed from the graphics part. In some cases the text strokes are misclassified as a graphical part, leaving them behind in the whole diagram. When text strokes are misclassified, this can be handled easily compared to when the graphical parts are misclassified as a text. This misclassification problem is explained in Section 5.2. During this phase, we used the nine FKI features proposed in Marti and Bunke [2001]. In this phase, BLSTM-RNN presented in Section 2.8.2.1 was

used. During the experiments, we used the standard network configuration used in Alvaro et al. [2014]. The configuration of the BLSTM-RNN classifier used is the same for all the experiments. The feature set determined the input layer size and the output layer size is the number of symbol classes. The forward and backward hidden layers comprise 100 LSTM memory blocks each. The weight of the network was initialised with a Gaussian distribution of mean 0 and standard derivation of 0.1. Online gradient descent with a learning rate of 0.0001 and a momentum of 0.9 was used to train the network. For more details regarding our experiment and the result of the evaluation performed on our dataset, please refer to Section 5.2. All the experiments were performed on our created datasets and the IAM Handwriting Database[1] described in Section 2.7.1.1 [Marti and Bunke, 2002]. These results are presented in Section 5.2 for both the biased and unbiased cases.

3.5 Stroke Processing

During this phase, the strokes of the FA diagrams are extracted and reconstructed as shown in Figure 3.1. Each pixel of the thinned FA is labelled using its 3×3 neighbourhood. The main reason for doing this is to find a trajectory of the stroke within a 3×3 window if it fits our predefined patterns. The algorithm we used here takes as input a matrix of pixels that represents the entire bitmap diagram.

3.6 Segmentation

Handwritten document image segmentation has generally been regarded as one of the most challenging tasks when developing a recognition system (see Section 2.4.5), especially when the system is based on an offline approach [Graves and Schmidhuber, 2009; Hu and Zanibbi, 2016]. Different segmentation algorithms have been proposed in Sonka et al. [2014]; Mahdavi et al. [2019]. However, most of these algorithms are not efficient enough for the purpose of this research because of hand-crafted features of strokes designed for segmentation may be sensitive to large variations of images. Also, to avoid errors accumulated from strokes been considered as noise, and some features of strokes been grouped into a wrong class, most of these existing algorithms use online recognition approaches.

As explained in Alvarado [2007], when developing a recognition system, it is important to focus the segmentation algorithm on the domain of interest. This will reduce the

[1] http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

error rate of the developed system since it is domain focused. To get the best result from the segmentation phase, we introduced a new segmentation algorithm in Section 5.4. This algorithm focuses on hand-drawn FA diagrams. The algorithm leverages on FA diagram characteristics, i.e. all FA diagram components are made up of strokes that are somehow connected, hence forming a closed structure of strokes. The algorithm we introduced in this research has proven to be efficient enough for the purpose of this research even when we introduced noise to some of the input images. The performance and the results obtained from our new segmentation method is a justification that the method is efficient. The results are presented in Section 5.4.

3.7 Arrow Recognition

Arrow detection and recognition is a common phase when recognising connected diagrams as illustrated in Figure 3.1. Most of the research that has been conducted in this field treats an arrow as a symbol that is found between two other symbols during recognition. In an FA diagram, there are three types of arrows, namely, the start arrow, the normal arrow and the loopback arrow. For an FA diagram, this approach will fail to recognise the start arrow and the loopback arrow because the start arrow only points to a single circle and the loopback arrow points back to the same circle where it starts from.

To achieve our aim, we made two assumptions. Firstly, most people draw an arrow from the shaft to the end, and secondly, arrows are usually drawn as two consecutive pen strokes and as a single pen stroke in a few cases. It is important to develop an algorithm that can efficiently interpret all the type of arrows in an FA diagram. To do this, we introduced an algorithm similar to that of Kara and Stahovich [2007]. The algorithm proposed in Kara and Stahovich [2007] was not efficient enough for this research. The algorithm only focuses on recognising single stroke arrows. Based on our assumption, this algorithm will fail when arrows drawn as two pen strokes are used to test the algorithm. We went further to extend this algorithm to recognise arrows drawn as two consecutive pen strokes. The Kara and Stahovich [2007] approach has been described in Section 2.5.4. The new approach is explained in Section 5.5.

3.8 Shape Detection and Recognition

Recognition of shapes found in diagrams such as flowcharts, FA diagrams, UML diagrams, and chemical structure diagrams is a common stage when proposing recognition systems for connected diagrams. The input to this phase is stroke clusters as shown in Figure 3.1. Some existing systems that have addressed this research area were described in Section 2.5 and 2.8. For this research, our focus is on an FA state shapes. At this stage, the data are state and accepting state symbols. After examining some existing classification algorithms, we decided to use support vector machines (SVMs). SVMs was the chosen algorithm because it has achieved state-of-the-art in classifying connected diagram shapes Ouyang and Davis [2011]. We used SVMs presented in Section 2.8.1 to classify FA states into the right class they belong to.

The visual and geometric features proposed in Zhang and Lu [2002] were used as the feature descriptor. This descriptor is presented in Section 2.5.4. This descriptor was chosen for this research because of its compatibility with shape representation and it is also computationally effective. Feature descriptors proposed in Ding et al. [2009]; Alvaro et al. [2014]; Hu and Zanibbi [2016] may have improved the performance of the shape recogniser, but the results obtained from using the descriptor proposed in Zhang and Lu [2002] was good enough for this research. The obtained results are presented in Section 5.6.

3.9 Text Detection and Recognition

Optical Character Recognition (OCR) is used to recognise all the removed text and all unused strokes. Numerous OCRs have been proposed in literature [Memon et al., 2020]. For this research, we used an open-source OCR known as Tesseract OCR [Smith, 2007]. We chose Tesseract among other OCRs because it is referred to as the most accurate open source OCR engine available to recognise text from images and scanned documents. This OCR is used to process the contents of the text bounding boxes obtained from Section 3.4. More details about this can be found in Section 5.7.

3.10 Performance Evaluation

The evaluation of the proposed approaches for hand-drawn FA diagrams was performed at the structure level. Structure evaluation was done at the stroke labeling level and the symbol segmentation and recognition level. We used these evaluation

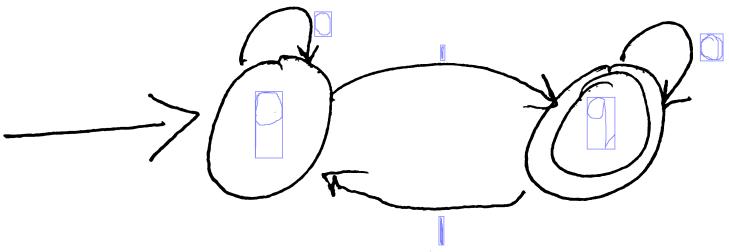


Figure 3.5: Detecting and recognising the text in Figure 3.3

levels because they have been used to evaluate most state-of-the-art work in handwriting recognition [Wu et al., 2015]. These evaluations were done to check if the output contains the same number of strokes and symbols as the ground truth, and also to check each stroke label against its ground truth in the database. All experiments were performed on the created databases. Comparisons were made with an existing online database of FA diagrams. The time taken for each of the recognition phases and the whole recognition system as a single unit was also reported. All of these are explained to details in Chapter 6.

3.11 Conclusion

This chapter presents the general methodology of the present work, with the aim of introducing a new recognition system that uses offline approaches for recognising hand-drawn FA diagrams. From Figure 3.1, most of the pre-processing steps are well-known approaches with only a little modification during the input image thinning process. We have created our dataset of hand-drawn FA images for this research. A new stroke extraction and segmentation algorithms were introduced in this research. During arrow recognition, we extended the existing Kara and Stahovich [2007] arrow detection and recognition algorithms for the purpose of our research.

The working process of the proposed approaches is clearly shown in Figure 3.1. Moreover, the phases involved in the research methodology are also discussed in each section of this chapter. The next chapter discusses the pre-processing and thinning techniques used on the collected data. As explained in the previous chapters, the main reason for pre-processing these images is to improve the image data and quality by getting rid of undesired noise and distortions. It also helps in enhancing image features that are needed for further processing. The chapter also explains the algorithm used in

thinning the FA diagrams. The FA images were thinned because we want to reduce the data amount of the input FA diagrams after conversion to binary images.

Chapter 4

Pre-processing of Hand-drawn FA Diagrams

4.1 Introduction

The previous chapter highlights and explains the research methodology of this thesis. This chapter presents the pre-processing and thinning of hand-drawn FA diagrams. The fundamentals of image pre-processing and thinning have been explained in Section 2.4. As explained in the previous chapters, pre-processing techniques are the first approach in a recognition system. The output of the pre-processing phase may be used as input for the thinning phase. The difficulty and effort of recognising the captured FA diagram depend on the capturing device, in this case, a mobile phone. The possibility of correct recognition depends on the captured image quality, lighting conditions, and the distance between the camera and the FA diagram. In such a scenario, pre-processing the captured image may give a suitable output for further processing during recognition. During this phase, image variations such as rotation, shape, distortion, and size are key issues to be addressed.

The task of transforming a paper-based document into a digital image using mobile phone cameras or scanners is known as document acquisition. During this process, noise, illumination and skew (a change in image position) could be introduced into the image. If these diagrams are to be used in the future, this noise must be removed from the diagram. In particular, any irrelevant bit patterns that have no importance (for example the shadow on the bottom right in Figure 4.1 is not needed during the FA diagram recognition) in the expected output must be removed. This makes pre-processing necessary in most document image analysis and classification tasks.

If the output of the input diagram after pre-processing meets the specifications of the further operation, then the further operation takes the pre-processed output as its own input. Algorithm 4.1 illustrates the different stages that an input image (FA) goes through during its pre-processing.

Algorithm 4.1: Preprocessing

1 **Input:** Digitized hand-drawn FA RGB image
2 **Output:** Preprocessed image
3 **Methodology**
4 Step 1: Read an RGB image
5 Step 2: Convert the image to a grayscale image
6 Step 3: Perform histogram equalisation
7 Step 4: Remove the noise using median filter
8 Step 5: Convert the grayscale image to a binary image using a threshold value
9 Step 6: Normalize the image to a predefined size of 1280×720 pixels
10 Step 7: Convert the normalized image to single-pixel thickness using thinning.

Figure 4.1 shows an RGB (red, green, and blue) hand-drawn FA image that will be used as the input image. The image size is 1280×720 pixels, i.e. 452×254 mm with a resolution of 72 dpi. This image is affected with a good deal of noise, i.e. a noise density of 0.09. This affects approximately 9% of the pixels in the image. To improve the image as much as possible for further processing, i.e. feature extraction and classification, different pre-processing operations will be performed on the image.

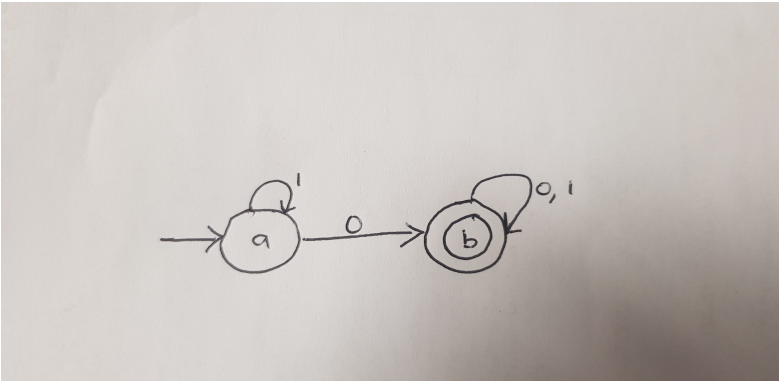


Figure 4.1: Input RGB FA diagram

The input images will be converted to grayscale images (Figure 4.2) because the colour information does not help us identify important features and edges. Grayscale (i.e. intensity) will be sufficient to identify these features.

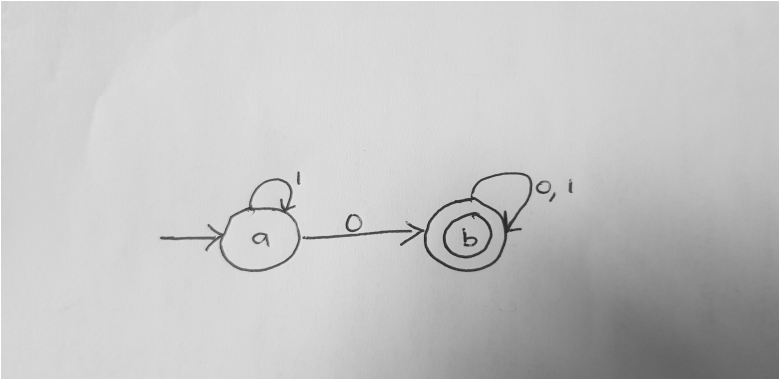


Figure 4.2: Input FA diagram after conversion to grayscale

4.2 Image Enhancement

During image acquisition, the quality of the image may be downgraded because of bad illumination, and motion blur. The aim of image enhancement is to make the output of a processed image more visually appealing and suitable for the next operation than the original image. In this work, FA images are resized because it is important that all the images have the same properties. Also, we adjusted the brightness and contrast of the images automatically using their histograms.

Histogram equalisation (see Algorithm 4.2) was performed on the images. Histogram equalisation is a method used to adjust the intensity of an image to enhance its contrast. The histogram equalisation is computed as follows [Wong et al., 2016]. Let C represent an input image with intensity levels (seen as random variables) in intervals $[0, ..., L-1]$, n represents the scale of intensity, and f_k denotes the number of pixels whose intensity magnitude is at the kth level. Let each element in C be a pixel and the values of the elements represent the intensities of the pixels.

The probability density function of the image C is given as:

$$p(C) = \frac{f_k}{n}. \qquad \text{(Equation 4.1)}$$

For example, consider the grayscale image in Figure 4.3.

$$C = \begin{array}{|c|c|c|c|} \hline 3 & 2 & 4 & 5 \\ \hline 7 & 7 & 8 & 2 \\ \hline 3 & 1 & 2 & 3 \\ \hline 5 & 4 & 6 & 7 \\ \hline \end{array}$$

Figure 4.3: Input image C

Algorithm 4.2: Histogram Equalization

1 **Input:** Grayscale image
2 **Output:** Enhanced image
3 **Methodology**
4 Step 1: Read a grayscale image
5 Step 2: Count the total number of pixels associated with each pixel intensity in the image
6 Step 3: Calculate the probability of each pixel intensity in the image matrix
7 Step 4: Calculate the cumulative probability (CP)
8 Step 5: Multiply CP by the scale intensity
9 Step 6: Round the decimal values obtained to the lower integer values.

In Figure 4.3, the intensity of the pixels is between 1 to 8. To expand or stretch the contrast of C, histogram equalisation using a scale intensity of 1 to 20 (this scale is used because C is dark and not legible and the focus is on increasing its contrast to make it legible) was used. This process is presented in the steps below. The corresponding outputs of each of these steps are presented in Table 4.1 –

Step 1: The total number of pixels associated with each pixel intensity is counted as shown in Step 1 in Table 4.1.

Step 2: The probability of each pixel intensity in C is calculated using Equation 4.1, where f_k is the number of pixels in Step 1 and $n = 16$.

Step 3: The cumulative probability (CP) is then calculated by adding the previous probability to the current probability under consideration in CP. For example, in Step 3 in Table 4.3, to get the CP of the pixel intensities 2, add $.0625 + .1875 = .2500$. To get the CP of the pixel intensities 3, add $.1875 + .2500 = .4375$. This process is continuous until the last pixel intensity (i.e. 8) is visited.

Step 4: Since the aim is to change the intensity range to 1 – 20, the cumulative probability is multiplied by 20.

Step 5: The final intensity is determined by rounding down the values to the lower or floor integer values.

As shown in Figure 4.4(b), the intensity range of the pixels has been increased and hence, the histogram of the image will look more even. Also, the image is enhanced for further processing.

Table 4.1: Histogram equalisation process of Figure 4.3

Step 1 – number of pixels in C

Pixel intensities	1	2	3	4	5	6	7	8
Number of pixels	1	3	3	2	2	1	3	1

Step 2 – pixel intensity probability

Probability	.0625	.1875	.1875	.1250	.1250	.0625	.1875	.0625

Step 3 – cumulative probability of the pixel intensities

Cumulative Probability (CP)	.0625	.2500	.4375	.5625	.6875	.7500	.9375	1

Step 4 – cumulative probability after scaling to 20

CP * 20	1.25	5	8.75	11.25	13.75	15	18.75	20

Step 5 – final pixel intensity

Final output	1	5	8	11	13	15	18	20

3	2	4	5
7	7	8	2
3	1	2	3
5	4	6	7

(a) Input image

$\rightarrow$

8	5	11	13
18	18	20	5
8	1	5	8
13	11	15	18

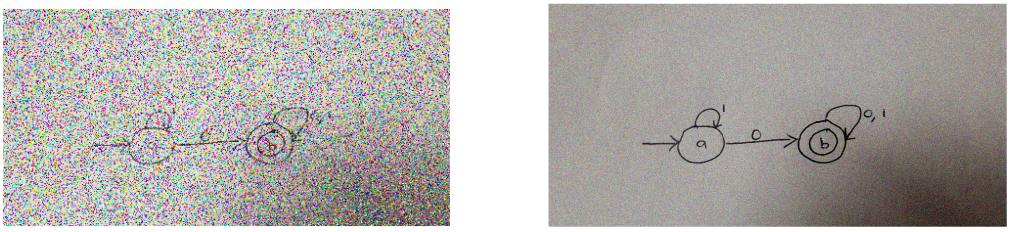
(b) Output of Figure 4.3 after equalisation

Figure 4.4: Final output after histogram equalisation

4.3 Noise Removal

As explained in Section 2.4.2, noise is a common problem encountered when capturing images. It can occur due to the type of capturing device used or the quality of the paper used. Noise can come up either in the background or foreground of an image; it can also be generated before, during and after capturing. In this work, a median filter (3×3 window and standard deviation $\sigma = 3.0$) was used to remove most of the noise from the input FA images. A median filter (explained in Section 2.4.3.2) is a non-linear digital filtering technique that uses statistics. It is sufficient to remove any type of noise found in the diagrams in our database because these diagrams are simply connected lines of strokes drawn on a white background. We chose this filtering technique because of the following:

- it is much more efficient than the averaging or mean filter at removing any type of noise especially the salt-and-pepper noise, and
- while removing this noise, its ability to preserve edges compared to other techniques is effective.



(a) FA diagram affected with salt-and-pepper noise

(b) FA diagram affected with Gaussian noise

Figure 4.5: Figure 4.2 affected with two types of noise

Figure 4.5(a) illustrates an FA with its pixels affected with salt-and-pepper noise. Figure 4.6 shows the output of Figure 4.5(a) after applying a median filter on the image.

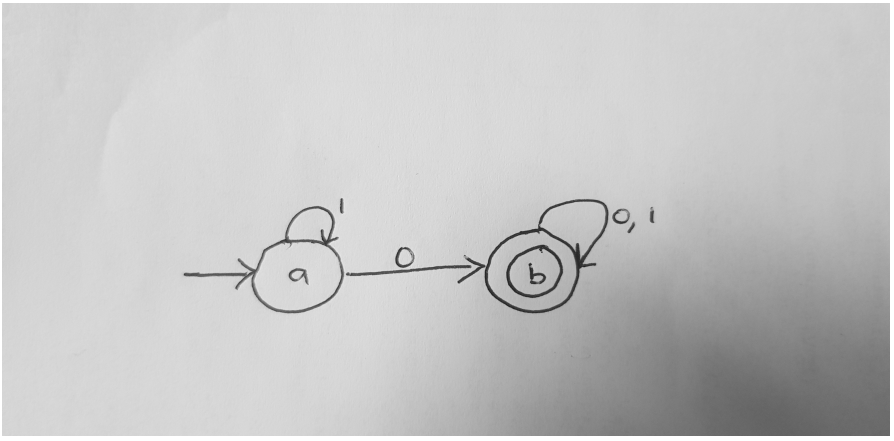


Figure 4.6: Figure 4.5(a) after applying median filter

A detailed explanation of applying a median filter with examples and illustrations was done in Section 2.4.3.2.

4.4 Thresholding

Thresholding involves extracting the ink (foreground) from the paper (background). This approach is used on gray images. The importance of thresholding and selecting

the right threshold has been explained in Section 2.4.4. Image thresholding can be done locally or globally. A global thresholding technique takes a single threshold value and uses it for the entire image. This differs from the local thresholding technique that considers the grayscale values of neighbouring pixels when determining the threshold value.

Global threshold t is a constant applicable over an entire image and is independent of pixel location and neighbour locations. Thresholding assigns a binary value of either 0 or 1 to each image pixel $f(x, y)$ according to threshold t.

$$g(x,y) = \begin{cases} 0 \text{ if } f(x,y) \leq t \\ 1 \text{ if } f(x,y) > t \end{cases} \qquad \text{(Equation 4.2)}$$

According to Babalola [2015], most diagrams like FAs, and UML diagrams are presented and drawn with strokes (black) that have the same tone and are well differentiated from the paper background (white). Therefore, global thresholding using Otsu's method (see Section 2.4.4 for Otsu's approach) will be suitable for processing FA diagrams. The major issue of global thresholding is selecting a good threshold value. Otsu's algorithm is the best approach to find an optimal threshold value. Otsu's method is seen as the benchmark and has been applied successfully in numerous studies [Moghaddam and Cheriet, 2012; Huang et al., 2012; Jianzhuang et al., 1991; Pan et al., 2016]. In this work, Otsu's approach will look for the threshold that minimises the intra-class variance (the variance within the class) in the input image, as shown in Algorithm 4.3 for two classes and their weighted sum of variances.

Algorithm 4.3: Global Thresholding

Data: Hand-drawn FA image B
Result: Thresholded FA image
1 initialize t;
2 segment B using t;
3 G_1 is the set of pixels brighter than t;
4 G_2 set of pixels darker than or equal to t;
5 compute the average of the intensities α_1 and α_2 of G_1 and G_2 as $\frac{\alpha_1 + \alpha_2}{2}$;
6 compute a new threshold value $t = \frac{1}{2}(\alpha_1 + \alpha_2)$;
7 repeat line 2 through 6 until the difference between values of t in successive iteration is smaller than a predefined parameter $\triangle t$.

Thresholding separates the FA image pixels aiming at making each cluster as tight as possible. In this work, Otsu's method was used to perform automatic image thresholding with an initial estimate threshold value of 155. Since all the images used in this research were drawn on a white paper, photographed using the sample mobile phone

and are all photographed under similar conditions such as illumination, the initial estimate value used (i.e. 155) works on all the images it was applied on. We checked the statistics of the pixels of the whole image and then we had two conditional probabilities, i.e. the mean and the variance of a class. This then separates the pixels of the image into two classes i.e. background and foreground. The output of this process is shown in Figure 4.7.

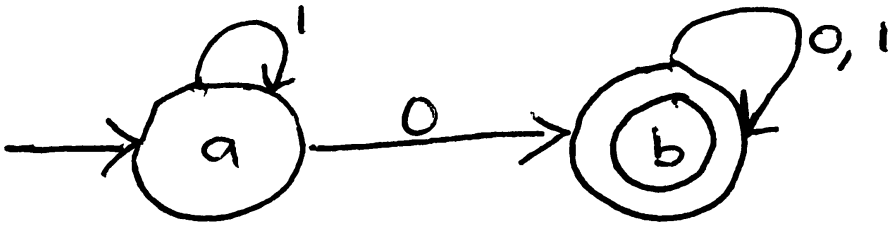


Figure 4.7: Thresholding of Figure 4.2

4.5 Thinning

The major reason for image thinning is to simplify the lines that make up the FA diagram into lines of one pixel wide. During this process, the structural information of the diagram such as direction, position and length of lines is retained. Initially, we adopted the Zhang-Suen algorithm explained in Section 2.4.6.1 [Zhang and Suen, 1984] during the thinning phase. After collecting data, we realised some students emphasise their FA drawing by making a line double or drawing a single line or a circle several times as illustrated in Figure 4.8. This affects the output of this phase, which in turn reduces the accuracy of the other recognition phases (for example, see the start arrows in Figure 4.9). So, we added more features to the existing Zhang-Suen system. These features improved the output of the thinning phase.



(a) Example of FA drawn by student A (b) Example of FA drawn by student B

Figure 4.8: Hand-drawn FAs with different emphasis

Although the Zhang-Suen algorithm has immunity to contour noise [Zhang and Suen, 1984] and is good at preserving diagram connectivity, there are still some drawbacks to this algorithm. Our focus is on FA diagrams, so we focus on improving only

the possible drawbacks encountered when recognising connected diagrams such as FAs, flowcharts and UML diagrams. When the erosion level in the diagram is too high, the algorithm is negatively affected [Deng et al., 2000]. This is because it erodes the white pixels in the boundaries of regions in a binary image. This makes the foreground pixels shrink in size and creates bigger holes in the image. When this algorithm is applied to such a diagram, the original structure of the diagram is partially lost. For example, the structure of the start arrow in Figure 4.8(a) will be lost during the thinning process because of its pixel size and the emphasises of the pen strokes. The process outputs an undesired start arrow as shown in its corresponding output in Figure 4.9(a). This result will affect the further processing of this diagram.



(a) Applying Zhang-Suen Algorithm on Figure 4.8(a)

(b) Applying Zhang-Suen Algorithm on Figure 4.8(b)

Figure 4.9: Zhang-Suen Algorithm on Figure 4.8

The approach described here is similar to the Zhang-Suen approach; however, we focus on considering more edge or border pixels by expanding the edge size and not creating holes in the output images like those in the start arrows in Figure 4.9.

Definition 4.1. A pixel p at coordinates (x, y) has four horizontal and vertical neighbours whose coordinates are given by: $(x+1, y), (x-1, y), (x, y+1), (x, y-1)$. This set of pixels, called the 4-neighbours of p, is denoted by $N_4(p)$ [Gonzalez et al., 2004].

Definition 4.2. The four diagonal neighbours of p have coordinates: $(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$ and are denoted by $N_D(p)$. These points together with the 4-neighbours are called 8-neighbours of p, denoted by $N_8(p)$ [Gonzalez et al., 2004].

The approach is based on two different techniques. All the conditions in the techniques are check during thinning. In both techniques, the pixel P_1 is the pixel under consideration and it is deleted (i.e. changed from 1 to 0) if it fulfils the following;

$A(P_1) = 1$ means that pixel P_1 has only one 8-connected component in its neighbourhood. This implies that P_1 can be deleted even when it is an edge pixel and the deletion will not affect its 3×3 neighbourhood. Conditions (*) in Technique 1 are used to ensure that no hole is created in the 3×3 window during thinning. The number of neighbouring black pixels for P_1 is at least 2 and not more than 7, i.e. $2 \leq I(P_1) \leq 7$ is used as opposed to $2 \leq I(P_1) \leq 6$ used in the Zhang-Suen approach. The number of

white-to-black transitions around P_1 in the order $P_2,\ P_3,\ P_4,\ P_5,\ P_6,\ P_7,\ P_8,\ P_9,\ P_2$ is represented as $A(P_1) = 1$, and $I(P_1)$ represents the number of black pixel neighbours of P_1, i.e. how many of the pixels from P_2 through P_9 are black.

Technique 1: The aim of the first step is to locate the pixel P_1 that fulfils the four conditions listed below. If a pixel satisfies the four conditions, it will be set to white. All pixels are tested and pixels satisfying all the following conditions are noted at this stage.
$A(P_1) = 1$
$2 \leq I(P_1) \leq 7$
$P_2 \times P_4 \times P_6 = 0$ (*)
$P_4 \times P_6 \times P_8 = 0$ (*)

Technique 2: All the pixels are tested again and pixels satisfying the conditions below are noted. The conditions are the same as for Technique 1, except that the last two conditions above are changed.
$A(P_1) = 1$
$1 \leq I(P_1) \leq 7$
$P_2 \times P_4 \times P_8 = 0$
$P_2 \times P_6 \times P_8 = 0$

We adapted the approach proposed in Boudaoud et al. [2015]. In this approach, one pixel (i.e. 7) was added to the edges of the input image as opposed to 6 that was used in Zhang and Suen [1984]. This is to make sure that more edges are considered during thinning. Also, it creates more space for the edge pixels to be used as padding pixels. Image padding is a technique used to introduce new pixels around the edges of an image. This will help in deleting more edge and border pixels.

The adapted approach starts its operation from the left side of the diagram, i.e. it finds and picks a point (this point is referred to as the target point) that belongs to the skeleton region for processing. Structuring elements starting from B_1 to B_8 are used to find this point in the diagram by placing the element over the image under consideration. After applying Techniques 1 and Technique 2 on Figure 4.8(a), we have Figure 4.10. The output is improved when compared to the existing one, Figure 4.10 is the final output of this phase. The image edges have been reduced when compared to the input image. This output is then used as an input to the next phase which is the extraction and reconstruction of the pre-processed image.

As stated earlier, the existing thinning algorithm was adapted because of the nature of our dataset, i.e. students tend to draw lines as a double line. Based on the results obtained from the existing algorithm, we can conclude that it would still work if the focus

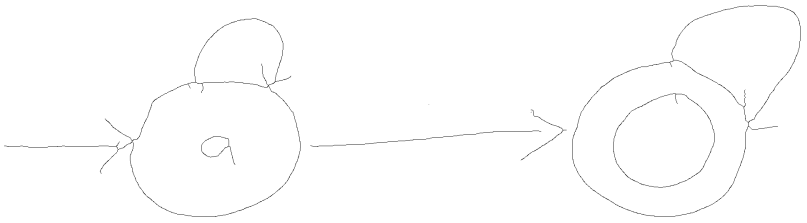


Figure 4.10: Output of the thinning algorithm

of this thesis were on online recognition or if the mode of collecting data was strict, i.e. students are instructed to be careful with their drawings and not emphasise strokes. The adapted algorithm has been tested on about 500 diagrams such as FAs, flowcharts and UML diagrams and gave better output on about 90% of the input images when compared to the Zhang-Suen algorithm.

4.6 Summary

This chapter discusses histogram equalisation, noise removal, thresholding and thinning which are all pre-processing techniques suitable for hand-drawn FA diagrams. Each of these techniques is applied to all the diagrams in the databases. Due to the writing style of some students, the accuracy of the existing thinning algorithm was low and failed in some thinning attempts. To improve the accuracy of the recognition system, we have improved on the existing Zhang-Suen thinning algorithm by making a two-pixel wide stroke a one-pixel wide stroke.

The next chapter discusses the recognition system from the stroke level through to the recognition level. After the input FA image has been pre-processed and thinned, the output of the thinning stage is used as input during the reconstruction of strokes. The chapter also explains the divide-and-conquer approach used during the separation of text from graphical components in an FA diagram. Furthermore, the recognition phase which was done after the image segmentation stage is explained. The recognition process was divided into three different phases, namely, arrow recognition, state shape recognition and text recognition. The accuracies of each of the phases will also be shown.

Chapter 5

From Stroke to Finite Automaton

5.1 Introduction

In the previous chapter, image pre-processing was explained with a focus on techniques suitable for FA diagrams and diagrams with a similar structure. The chapter also shows the process of thinning the FA diagrams using the improved thinning algorithm which gives a better output compared to the existing thinning algorithm. Where the existing algorithm failed in our study was also highlighted.

This chapter focuses on how to extract the features and reconstruct and classify the strokes from the pre-processed FA diagrams. As explained in Chapter 2, many existing approaches to recognise hand-writing and hand-drawing would have given a better result if there were possibilities of guessing accurately the original movement of the pen that was used by the user to write a character or draw a diagram.

5.2 Text and Graphics Separation

After observing that the textual component bears little information about the whole diagram structure during image processing, we decided to separate this component from the graphical component. Text and graphics separation is a common approach used during document image analysis [Hoang and Tabbone, 2010; Bresler et al., 2015]. One of the main reasons for using this divide-and-conquer approach is to reduce the complexity of the system since the number of strokes in the diagram will be reduced after removing the textual components. Basically, during this phase, all the text strokes are taken out of the whole diagram, and the whole diagram is recognised without the text. Text strokes are replaced after the graphical components have been recognised.

There exist some classifiers [Hoang and Tabbone, 2010; Bishop et al., 2004; Bresler et al., 2015] that can perform the task of separating text from graphics. These classifiers will work well on detecting text. However, they might not be able to classify the graphical component of an FA accurately due to the domain the classifier was developed for (i.e. the training dataset used in implementing the classifiers). For example, a classifier trained with flowchart diagrams will have a higher accuracy when it is used to recognise flowchart diagrams compared to when the same classifier is adopted for the purpose of recognising FA diagrams. This is because the classifier knows what the different components of a flowchart diagram (i.e. decision, input and output, process) look like and does not understand that of an FA diagram (i.e. start state and final state). This led to the development of our own classifier as explained below. This classifier was trained only for FA diagrams.

The text and graphics separation algorithm aim at classifying each single stroke into class text or class graphic. To achieve this, we used the text and graphics separation algorithm proposed in Li et al. [2018]. In this algorithm, the unary neural network (UNN) and pairwise neural network (PNN) are both formulated as multilayer perceptrons. The unary features extracted from each single component of the image are taken as input by the UNN and the concatenation of pairs of these unary features are the input to the PNN. Detection and recognition of FA text strokes are explained in Section 5.7. We then used a bidirectional long short-term memory recurrent neural network (BLSTM-RNN) to capture the global context. This type of neural network joins two hidden layers of opposite directions to the same output, thus enabling the network to access context information in both time directions. One hidden layer processes the input sequence forward, while the other hidden layer processes it backward. The output layer sums up the activation levels from the two hidden layers at each position in the text. The output activation of the nodes in the input layer is then normalised, to sum up to 1. Hence, they can be treated as a vector showing the probability for each label to occur at a specific position. This network has been used and has outperformed standard classifiers like Hidden Markov models and recurrent neural networks in the recognition of symbols and handwriting [Alvaro et al., 2014].

In this work, the FKI features proposed in Marti and Bunke [2001] were used. These geometric features were used because it has achieved state-of-the-art results in handwriting literature. Given a binary image of height H, the following nine geometrical values c can be computed for each column a:

- Number of black pixels in the column:
  $c_1(x) = \sum_{y=1}^{H} I(x, y)$

- Center of gravity of the column:
  $c_2(x) = \frac{1}{H} \sum_{y=1}^{H} y.I(x,y)$
- Second order moment of the column:
  $c_3(x) = \frac{1}{H^2} \sum_{y=1}^{H} y^2.I(x,y)$
- Position of the upper contour in the column:
  $c_4(x) = min\{y \mid I(x,y) = black\}$
- Position of the lower contour in the column:
  $c_5(x) = max\{y \mid I(x,y) = black\}$
- Orientation of the upper contour in the column:
  $c_6(x) = \frac{c_4(x+1) - c_4(x-1)}{2}$
- Orientation of the lower contour in the column:
  $c_7(x) = \frac{c_5(x+1) - c_5(x-1)}{2}$
- Number of black-white transitions in the column:
  $c_8(x) = NT_{black \to white}\ I(x,y); 1 \leq y \leq H$
- Number of black pixels between the upper and lower contours:

$$c_9(x) = \sum_{c_4(x) < y < c_5(x)}^{H} I(x,y)$$

Lastly, if we are given a binary image of size $W \times H$, for each column x such that $1 \leq x \leq W$, this will output a 9-dimensional feature vector $f_{fki}(x)$ as:

$$f_{fki}(x) = [c_1, c_2, ..., c_9] \qquad \text{(Equation 5.1)}$$

The network was trained by giving it an input sequence s made up of n vectors such that $s = (q_1, q_2, ..., q_n)$. The network then computes for each class c the posterior probability $P(c \mid q_i)$, where q_i is a feature vector. The posterior probability per symbol was derived using the normalisation as in Equation 5.2.

$$P(c \mid s) = \frac{1}{n} \sum_{i=1}^{n} P(c \mid q_i) \qquad \text{(Equation 5.2)}$$

where the probability per symbol for c is computed as its average probability per image. During the experiments, we used the standard network configuration used in Alvaro et al. [2014]. The configuration of the BLSTM-RNN classifier used is the same

for all the experiments. The feature set determined the input layer size and the output layer size is the number of symbol classes. The forward and backward hidden layers comprise 100 LSTM memory blocks each. The weight of the network was initialised with a Gaussian distribution of mean 0 and standard derivation of 0.1. Online gradient descent with a learning rate of 0.0001 and a momentum of 0.9 was used to train the network. This network configuration has output good results in handwritten text recognition [Graves et al., 2009; Tolosana et al., 2018]. We tested our classifier on the IAM Handwriting Database[1] [Marti and Bunke, 2002], which is a database for English sentences developed for offline handwriting recognition. The database contains 5685 English sentences written by 657 writers under no constraints in the style of writing (see Figure 5.1). All text is written on a white paper and scanned at 300dpi resolution. The classifier achieved an accuracy of 96.5% on the database. It is important to say that some image recognition tasks such as labelling and segmentation have been done already on the IAM database by the authors of the database.

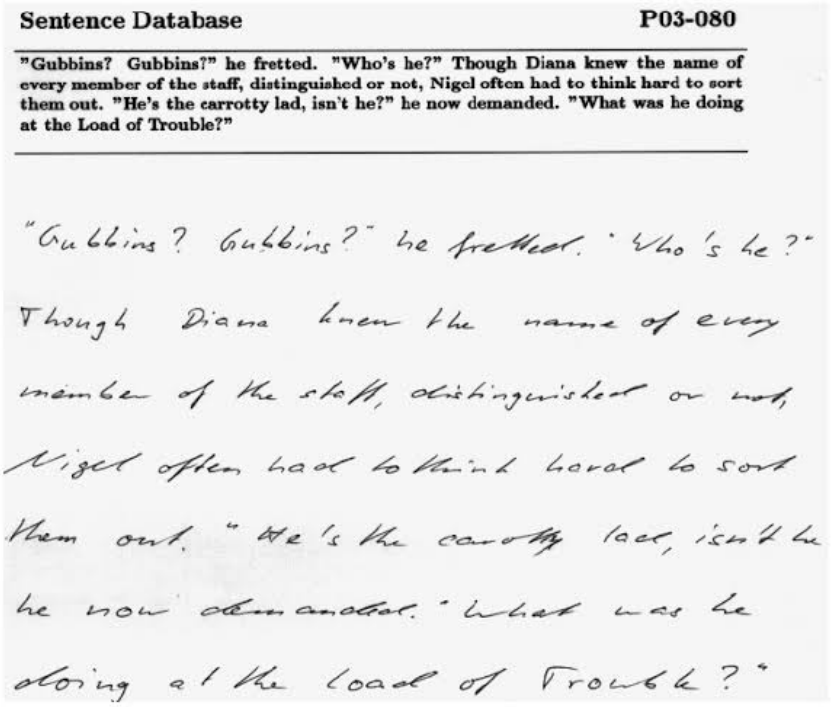


Figure 5.1: Sample handwritten document from the IAM handwriting database

Note that DB_a and DB_b stand for synthetic data database and raw data database respectively (see Section 3.2 and 6.2). We biased the result of the classifier for an FA diagram. This is to ensure that the classifier is certain that a stroke is a text before marking it as text. The classifier achieved a precision of 98.5% on the DB_a dataset, i.e. 96.7% and 97.6% for the classes shape and text respectively, and an accuracy of 86.4% on the DB_b dataset, i.e. 89.2% for class shapes and 83.5% for class text in an unbiased case.

[1] http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

For the biased case, the classifier achieved an accuracy of 97.2% on the DB_a dataset, i.e. 96.8% for the class shapes and 97.5% for the class text, and an accuracy of 85.4% on the DB_b dataset, i.e. 84.6% for the class shapes and 86.2% for the class text.

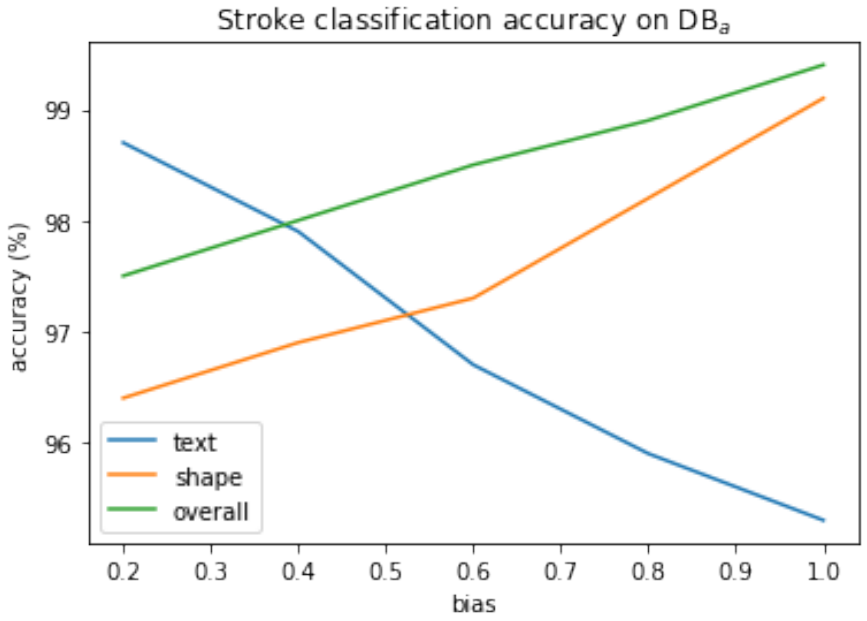


Figure 5.2: Stroke classification accuracy for the DB_a images in relation to bias

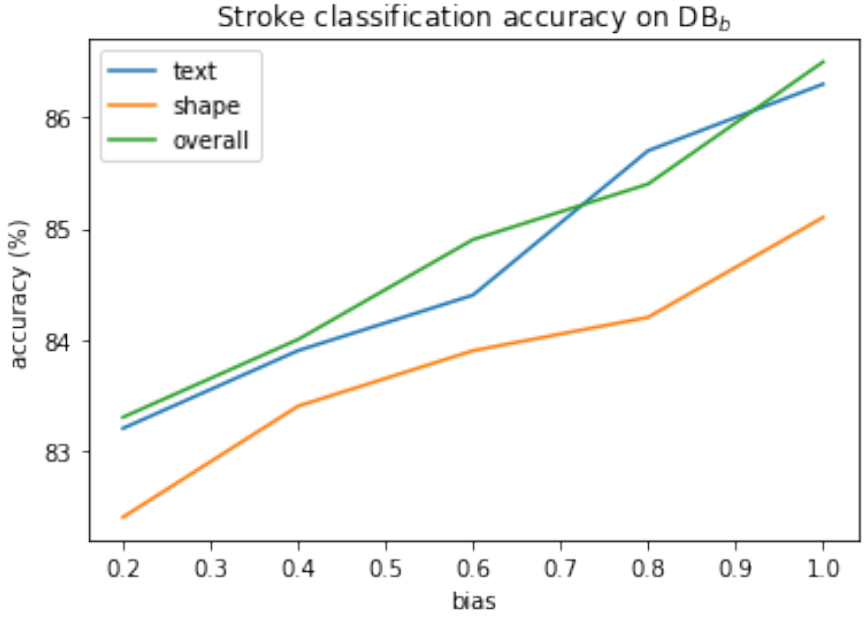
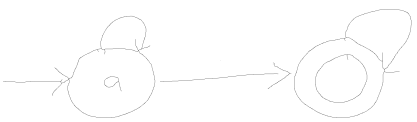


Figure 5.3: Stroke classification accuracy for the DB_b images in relation to bias

5.3 Stroke Processing

The tasks involved in processing FA strokes include the extraction of the strokes and labelling of the strokes. The output of this phase will be used as input during the classification and recognition phases. The main reason for reconstruction is to find a trajectory of the stroke within a 3×3 window if it fits our pre-defined patterns of FA diagrams.

The input to the phase is a pre-processed (thinned) FA diagram (see Figures 5.4(a) and 5.4(c)) and the expected output is a group of extracted and reconstructed strokes that makes up the input diagram (see Figures 5.4(b) and 5.4(d)). A stroke can be described as a sequence of points noted during a pen movement (i.e. between pen-down and pen-up).



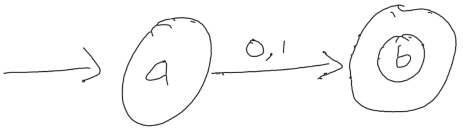
(a) A thinned FA diagram used as input



(b) Reconstructed FA diagram



(c) A thinned FA diagram used as input



(d) Reconstructed FA diagram

Figure 5.4: Thinned FA diagrams and their corresponding reconstructed strokes

After the input image has been pre-processed and converted to a width of one pixel during thinning, we extract and distinguish each stroke that is different. Strokes are said to be different if they are not drawn or written during one pen movement. The input used during the extraction process is a matrix of pixels. The matrix represents the whole thinned input bitmap FA diagram. It is made up of label 0's and label 1's. The extraction process is explained using the following steps:

- The algorithm starts by scanning the thinned input image from the topmost left to the bottom right (see Figure 5.5) and picks the first stroke it detects. Once a stroke is found, we pick the first point or pixel identified as 1 (foreground) on the stroke. This point is called the starting point.
- The first point is now known and picked to belong in the first stroke. This point is saved in a list in the stroke database. The stroke database is used to keep the

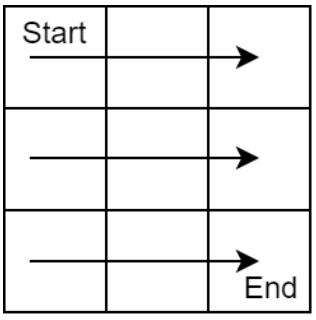


Figure 5.5: Scan direction

record of the pixels classified as part of the stroke under consideration. Stroke pixels can be stored as a list and retrieved from this database.

- Since we are using binarised FA images, points labelled as 1 (foreground) are visible to the algorithm, and those labelled as 0 are invisible. This is because the background (0) does not form a stroke, so we do not need it. Once a pixel with label 1 has been visited and added to the pixel list in the database, the algorithm changes this pixel label to 0, making it invisible. This is done to ensure that a stroke is visited only once and also to make the system faster and more efficient.

- We trace the stroke using the pixel found in the starting point. To select the next pixel to be added to the stroke database, we check for the 8-pixel neighbourhood around the current pixel. If the 8-pixel neighbourhood around the current pixel is empty except for one neighbouring pixel, we move to that pixel for processing. However, if there exists more than one pixel in the neighbourhood, the algorithm must determine which of the pixels to visit next. This process is explained later in this section. Once the single neighbouring pixel has been visited, it is added to the list in the stroke database. This pixel label is then changed from 1 to 0. Figure 5.6 illustrates an arrowhead pixels that shows the transition when there is only one choice available. The pixel in the red rectangular box in Figure 5.6 will be marked as visited by the algorithm before transiting to the next pixel. Algorithm 5.1 shows the extraction process when there is only one neighbour available.

Figure 5.7(a) illustrates an arrowhead used as the input image I. Since the input image is a thinned binary image, the pixels with letters (i.e. $A, B, C, ..., E$) in the figure are labelled as 1 and the pixels with 0 are labelled as 0. As explained earlier, the algorithm starts scanning through the whole input image from the topmost left pixel to the bottom right. The first pixel with label 1 that the algorithm detects while scanning from left to right will be chosen as the starting pixel, i.e. point A in Figure 5.7(b). The point or location of A will be identified as the starting point. Once A has been identified and

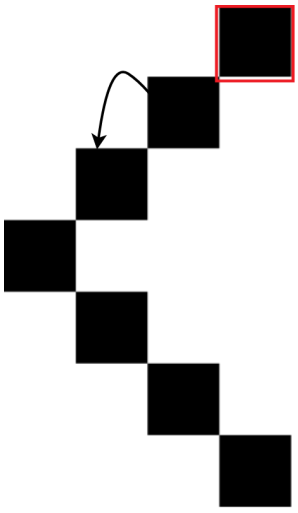


Figure 5.6: The transition when just one pixel is available

visited, its label changes from 1 to 0. The pixel A will now be added to the list in the database as illustrated in Figure 5.7(b) before moving to the next pixel with label 1, i.e. B in Figure 5.7(c).

Once the starting point has been identified, the algorithm aims at identifying all other pixels that make up the stroke under consideration. To do this, we introduced a pixel selector. The pixel selector chooses the next pixel to be added to the list in the stroke database by looking at the global features of the stroke being extracted at the moment. To determine the global direction, we look at a string of eight previous pixels instead of only looking at the single-pixel neighbourhood or the local direction. From the identified starting point A in Figure 5.7(a), the pixel selector is used to trace and identify all the pixels in the input image while storing each identified pixel in the database. The process of tracing, identifying and adding these stroke pixels to the list is continuous until the last pixel E is identified and added to the list, i.e. Figure 5.7(f). After all the pixels in the pre-processed image have been identified and added to the list in the database (see Figure 5.7 for illustration), the database now contains all the extracted stroke pixels of the input image. Since the current pixel, E is the last (i.e. no more pixels can be added to the list) pixel with label 1, the algorithm saves the list of noted points or pixels and considers the list as a stroke (i.e. the list in Figure 5.7(f)). The algorithm then discards the input and returns to the matrix. At this stage, the algorithm returns a stroke that consists of a list of pixels. If there exists more than a stroke in the image, the algorithm moves to the next available stroke to perform the same process and finally outputs all the strokes in the image.

Since the approach used in the extraction iterates from the left of the image matrix to the right, there is a possibility that the search and iteration for pixel points might get to a point where strokes may have more than one neighbour, i.e. where strokes overlap

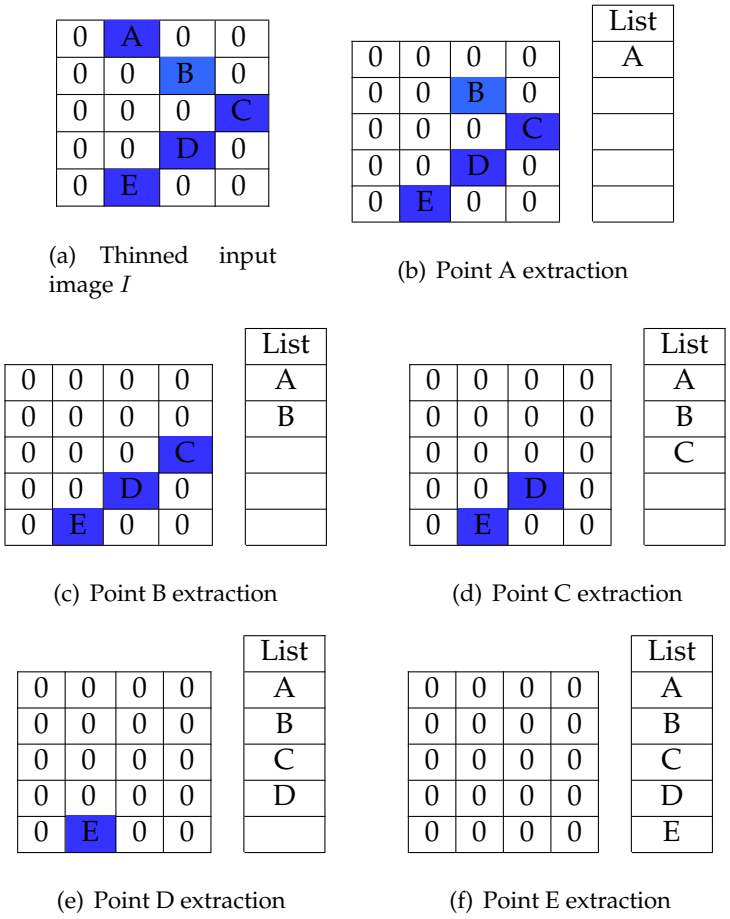
Algorithm 5.1: Stroke Extraction Process

Data: Thinned Hand-drawn FA Image I
Result: Extracted Strokes of I

```
1  I foreground are labeled as 1;
2  I background are labeled as 0;
3  scan through I from left to right ▷ find the first stroke pixel in I by
   scanning from the topmost left of the image to the right;
4  locate the first stroke pixel μ with label 1 found in I during scanning;
5  while μ exists in I do
6      mark μ as the starting point
7      if μ has been identified as the starting point and visited then
8          change μ label from 1 to 0
9          save μ in the list of the stroke database
10         discard μ and locate the next available pixel
11         using the pixel selector, identify and visit the next pixel to μ with label 1 in I
           for processing ▷ this process is repeated by scanning
           through the pixels of I and changing their labels from 1
           to 0 after visiting the pixels
12         if all points in I with label 1 have been identified and visited once, change their
           labels from 1 to 0 then
13             stored all these points in the list
14             trace the strokes from the starting point and extract them
15         end
16     end
17     discard I ▷ remove the image I and input another image for
       processing
18 end
```

or a branching point occurs along the stroke. In such a scenario, the algorithm must determine the pixel to visit next in order to continue the stroke. Once such a case is found, the algorithm creates another list for points to be ignored in the matrix. These ignored points are pixels with more than one neighbour, i.e. P in Figure 5.8. Once the iteration begins, the algorithm starts from point Y in Figure 5.8 and changes its label from 1 to 0 after visiting it before moving to point P. Point P has two neighbours Q and W. The algorithm must determine which of the pixels must be visited next. To do this, the algorithm retains the original location and value of the pixel P in the matrix and skips it (as illustrated in Figure 5.8(c)) before moving to the next available pixel by iterating from left to right. The pixel P is left out and visited after the algorithm has completed the iteration and the endpoint of the stroke has been detected, i.e. point W in Figure 5.8(c) has been identified.

After P have been skipped, point Q will be the first point with label 1 the algorithm will come across in the matrix during iteration from left to right. It is important to check the relationship that exists between points P and Q during iteration i.e. whether they



(a) Thinned input image I

(b) Point A extraction

(c) Point B extraction

(d) Point C extraction

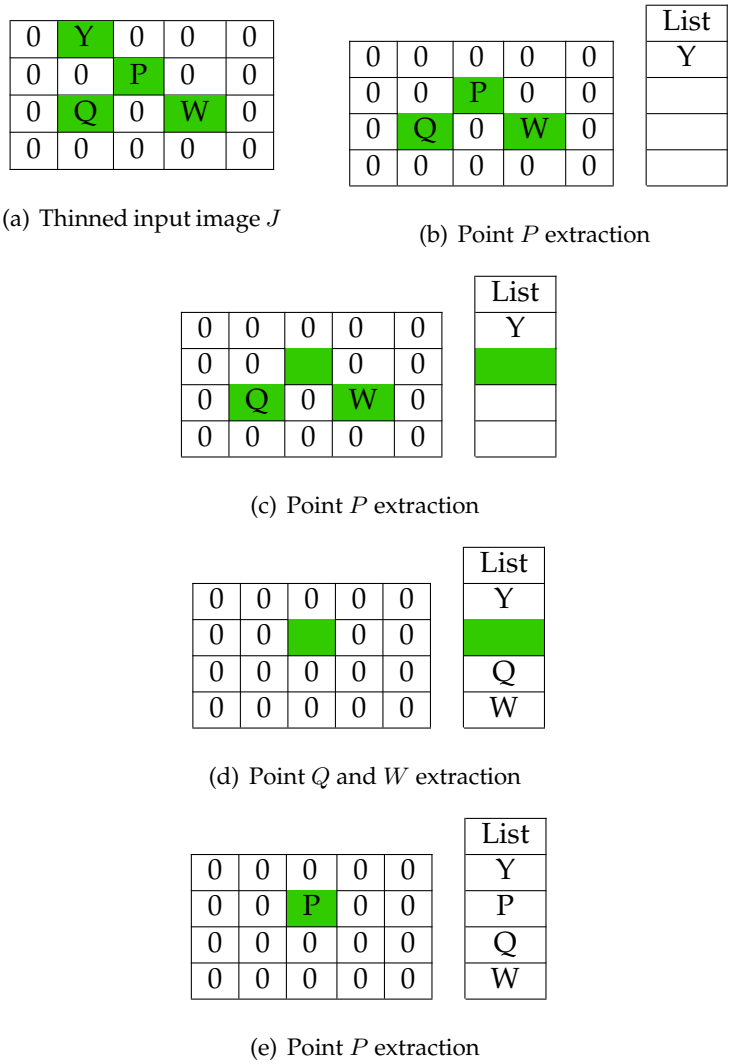
(e) Point D extraction

(f) Point E extraction

Figure 5.7: Example of extraction when only one pixel is available

are part of the same stroke or whether they are points of two different strokes touching or intercepting each other. If this check is not done, a pixel point in a stroke might be lost to another stroke. This will cause incorrect recognition and reduces the system accuracy during the classification stage. To perform this check, we check if the stroke under consideration (i.e. point P in Figure 5.8(b)) was drawn as a single pen movement with the branching pixel Q. If it was, the algorithm takes the two points to belong to the same stroke. If not, the iteration process skips point Q to point W and performs the same operation. Point Q will then be labelled as a pixel point in another stroke and visited when the stroke it belongs to is undergoing the extraction process.

If points P and Q belong to the same stroke, point Q will then be marked as visited and saved in the list. This iteration process is continued until all the pixels with label 1 have been identified and visited. Figure 5.8(d) illustrates the fully extracted pixels of J. However, the ignored pixel P is still in the stroke. To visit the pixel P again, the algorithm uses the single neighbour approach discussed earlier. The algorithm takes the entire pixels saved in the list (i.e. Y, Q, W) as a single entity. After this, the algorithm adds P to its original position in the list as illustrated in Figure 5.8(e) before returning



(a) Thinned input image J

(b) Point P extraction

(c) Point P extraction

(d) Point Q and W extraction

(e) Point P extraction

Figure 5.8: Pixel P with more than one neighbour

the list as a stroke. Point P will be tagged among the pixels in the list as a pixel with two neighbours during extraction. This process is explained with Algorithm 5.2.

To extract circles, we used the same approach as explained above. Since a circle is a closed structure, they are generally referred to as structures with no distinct or uncertain start and endpoints. Let us assume the circles in Figure 5.9 are represented in the matrix in Figure 5.10.

During iteration, if a closed symbol stroke is found, the algorithm assumes that this symbol is a circle because a circle is the only closed symbol component of an FA diagram. Once the algorithm identifies this symbol to be a circle, it picks the top-most pixel on the circle as the starting point, i.e. point A in Figure 5.10(a). Since the input images or the circle strokes under consideration are thinned already, the circles are made up of single pixels.

Algorithm 5.2: Stroke extraction process with a point having two neighbours

Data: Thinned Hand-drawn FA Image J
Result: Extracted Strokes of I

```
1  I foreground are labeled as 1;
2  I background are labeled as 0;
3  scan through J from left to right ▷ find the first stroke pixel in J by
   scanning from the top most left of the image to the right;
4  locate the first stroke pixel μ with label 1 found in J during scanning;
5  while μ exists in J do
6      mark μ as the starting point
7      if μ has been identified as the starting point and visited then
8          change μ label from 1 to 0
9          save μ in the list in the stroke database
10         discard μ and locate the next available pixel
11         using the pixel selector, identify and visit the next pixel (δ) to μ with label 1
           in J for processing
12         if the next pixel δ has two neighbour pixels say σ and τ then
13             skip δ and determine which of σ and τ to visit first
14             continue scanning from left to right
15             if σ is the first neighbour of δ found during scanning then
16                 pick σ as the next pixel to be visited
17                 change σ label from 1 to 0 and save it in the list
18                 discard σ and locate the next available pixel
19                 if all points in J with label 1 have been visited once and stored in the list
                   then
20                     go back to δ
21                     mark δ as visited using the approach explained in Algorithm 5.1
22                     trace the strokes from the starting point and extract the strokes of
                       J
23                 end
24             end
25             discard J    ▷ remove the image J and input another image
               for processing
26         end
27     end
28 end
```

Once the first topmost pixel (point A) has been identified as the starting point on the circle and added to the list in the database after changing its pixel value from 1 to 0, we explore the edges of the circle in an anticlockwise direction by moving across its pixels one after the other and adding them to the list after visiting them, i.e. move from point A to B and then point C until we get back to point A in Figure 5.10(a). Since it is a thinned circle, each of the circle pixels will have only one neighbour. Similar to the one neighbour pixel approach explained earlier in this section, once a pixel is found on the edge of the circle and marked as visited, its label changes from 1 to 0 before moving to the next available pixel and stops moving until the last pixel has been



(a) A complete circle

(b) An incomplete circle

Figure 5.9: Circle extraction

0	0	A	0	0
0	B	0	H	0
C	0	0	0	G
0	D	0	F	0
0	0	E	0	0

(a) Representation of Figure 5.9(a)

0	0	A	0	0
0	B	0	0	0
C	0	0	0	G
0	D	0	F	0
0	0	E	0	0

(b) Representation of Figure 5.9(b)

Figure 5.10: Circle extraction process

visited and changed from 1 to 0, i.e. point H in Figure 5.10(a). The algorithm moves from the identified starting point using the direction of the pixel connectivity similar to the chaincode approach [Kurnia et al., 2017] explained in Section 2.4.7. Once the extraction process is completed, the database will contain pixels of FA strokes that are to be recognised. It is important to segment the extracted stroke into segments after extraction. This will make differentiating the different components of an FA diagram easy.

5.4 Segmentation

The process of partitioning the extracted features in the FA diagram into numerous segments is done at this stage. Segmentation is a primary step for further image analysis and other computer vision problems. To the best of our knowledge, this research is the first to segment offline FA diagrams. Segmentation of hand-drawn connected diagrams is one of the most challenging tasks in image processing [Sonka et al., 2014]. The accuracy of the segmentation phase decides the eventual success or failure of the computerised analysis procedure. For this reason, considerable care should be taken to improve the probability of accurate segmentation. During segmentation, the output does not choose the components that really matches the whole pattern in the domain, i.e. final decisions are not made. The segmentation process only finds the groups of strokes that might form a symbol and determines in which possible relationships the components of the diagram domain can be to each other.

Segmenting free-hand drawn images automatically is still an open problem under active research [Mahdavi et al., 2019] and a challenging task, because heuristic relations or hand-crafted features of the strokes designed for segmentation may be sensitive to large variations of images. As explained in Chapter 2, many methods have been proposed for segmentation of diagrams. To avoid the accumulation of recognition errors, most of these methods aim at developing an online recognition system for diagrams [Wang et al., 2019]. Some other method assumes that each stroke represents a single shape, such as a single line or arc segment [Hu and Zanibbi, 2016]. Thus requiring the user to explicitly specify the intended segments. The single shape approach will work for circles but will fail to accurately segment arrows drawn as a two stroke (i.e. shaft separated from arrowhead) accurately. This makes these approaches not suitable for our research. Other work leveraged on using multiple information like the diagram colour and intensity during image segmentation. However, our datasets (i.e. black and white FA images) lack such information. So using such an approach will fail in this research.

In spite of the continuous efforts in the last decade, most of the work in hand-drawn image segmentation still mainly depend on temporal information of strokes [Wu et al., 2015]. Also, unlike our datasets which mainly contain rough and less carefully drawn diagrams and thus have FA diagrams with their components overlapped to each other in the image space, most available datasets in literature used in handwriting recognition typically contained non-overlapping components that are perfectly linked to each other. Therefore, basic methods that involve ordinary segmentation methods such as connected components analysis [Kovalevsky, 2019], geometric description [Deniziak and Michno, 2016], and projection histogram analysis [Ptak et al., 2017] will be sufficient for all these methods and inadequate for our problem. In this work, we explore a new approach using bounding boxes to segment free hand-drawn images. We focus on FA diagrams, without any semantic segmentation information as input and aim at getting semantically meaningful FA component-level structures.

A symbol is said to be correctly segmented (i.e. the result of the segmentation phase) if the following basis exists:

- strokes corresponding to the symbol are exactly the same as that in the ground truth, and
- label of the symbol is correct.

The basis listed above has been adopted in object detection literature and most state-of-the-art work in sketch parsing, hence we based our system evaluation on it. As

stated earlier, we used the bounding box instead of the stroke level [Wu et al., 2015] that has been used in most of the existing literature.



Figure 5.11: Stroke level segmentation approach

The main reason for using the bounding box is that sometimes text strokes may overlap or touch a non-text stroke; this will lead to incorrect segmentation at the stroke level but will be considered as correct in the bounding box.

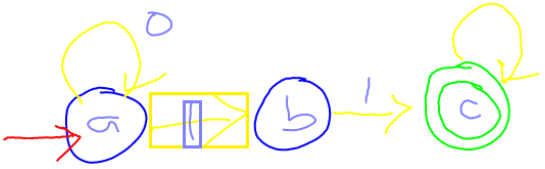


Figure 5.12: Bounding box segmentation approach

For example, consider the FA diagrams in Figures 5.11 and 5.12; both figures are the same FA diagram to be segmented. If we apply stroke level segmentation on Figure 5.11, then the label 1 intersecting the arrow between state a and state b in the FA will be classified as an arrow in this segmentation approach. However, the label 1 that intersects the arrow between state a and state b in Figure 5.12 will be classified as a text because it is segmented using the bounding box approach. Using the bounding box approach will eventually lead to having more FA diagram components recognised accurately and reduce the error rate during classification.

During the segmentation phase, the characteristics of FA diagram components (all FA diagram components are made up of strokes that are somehow connected to each other, hence forming a closed structure of strokes) are utilised. During this phase, we do not think of any specific shape a group of strokes form because we are only interested in the individual strokes that form the diagram and not the structure of the diagram, i.e. the connection and relationship between strokes were not considered here. We focus on

finding the distance and convexity between the endpoints of consecutive strokes that form the FA structure. The values from the distance and convexity are used to reject improper strokes in the structure and to compute the confidence of promising strokes. The process of detecting the strokes that form the structure is iterative and explained below using Figure 5.13.

Step 1: From the database of extracted strokes in Section 5.3, pick the initial single strokes from the diagram to be segmented. This step will generate the start arrow shaft stroke AS, the start arrowhead stroke AH and other strokes in Figure 5.13.

Step 2: Define the start and end-points of these strokes. In Figure 5.13, points 1 and 2 will be defined for stroke AS. Points 3 and 4 will be defined for the start arrowhead stroke AH. This process will be done for all the strokes in the FA diagram.

Step 3: Since the first stroke has been generated in step 1 above (i.e. AS), the second or next generated stroke (i.e. AH) is added to the first or previously generated stroke AS. This is done by looking at the proximity or closeness of the last generated stroke with that of the stroke under consideration. If they are close enough to each other to be tagged as an FA component, a bounding box is drawn around them. For example, in Figure 5.13, the arrow shaft stroke is the first generated stroke. The arrowhead stroke is added to the shaft stroke because it is close enough to it and was generated after it. This step is aimed at forming a start arrow symbol.

Step 4: In further iterations, strokes that can be added to unfinished structures are searched for in the generated strokes and added to the unfinished structure using the bounding box approach. For example, in Figure 5.13, the start arrow has been generated already using Step 1 to Step 3 above. The next FA component to be added will be the start state circle stroke and then the loopback arrow shaft stroke before adding the loopback arrowhead stroke. These steps are aimed at forming an FA diagram.

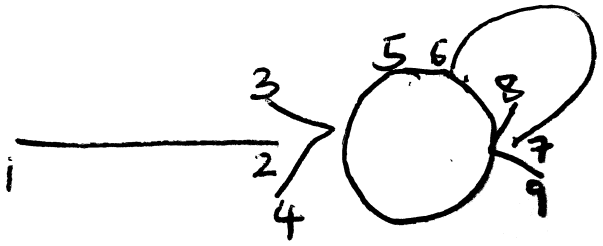


Figure 5.13: FA diagram to be segmented

The strokes to be added to the structure must fulfil the following criteria;

- the new stroke to be added must have one of its endpoints close to the last point of the unfinished structure. A stroke is said to be close to another stroke if the distance between them is smaller than a certain threshold $DT = \beta.D_{med}$, where the adaptive Distance Threshold is represented as DT, the median of the bounding boxes lengths of all single strokes in the diagram is D_{med}, and β is a constant that has been empirically defined to be 0.4 in this research. The reason for choosing 0.4 for β is based on the dependencies illustrated in Figure 5.14. This step is necessary because different handwriting and different photograph or scanning devices cause different scales. The β coefficient is indicated in Figure 5.14 by the green straight line. In Figure 5.13, the arrowhead is close to point 2 of the arrow shaft. That is why they can be joined together to form a component. This process is continued until the last stroke has been added to the unfinished structure. When this occurs, then the structure is closed and the average distance between successive strokes in the structure l_1 is computed.
- the violation of convexity is calculated as the maximal distance l_2 of any point from the convex hull. If the distance l_2 is more than the DT, the stroke is discarded immediately.
- all unfinished or undecided groups of strokes are discarded if they have over two strokes because no FA component can have over three strokes (drawing a circle is a single pen event, drawing an arrow can be done by one or two pen events).

These criteria are used to reduce the stroke search space. Equation 5.3 is used to compute the final confidence of the structure.

$$\text{conf}_{loop} = 1.0 - \frac{l_1 + l_2}{2.DT} \qquad \text{(Equation 5.3)}$$

As previously mentioned, segmentation is an important stage in image recognition. It is therefore important to reduce the error rate during segmentation to as low as possible, in order to have a good result during recognition. Diagram components splitting or division into two shapes is a common occurrence and error during segmentation. When a component is divided, such a component is incomplete. A diagram component is said to be incomplete if its features such as geometric and visual features are not complete. Also, if a component is not segmented as it appears in the ground truth, the component can be said to be incomplete. When this type of error occurs, the divided components are split into sub-shapes. For example, consider the state shape A of an

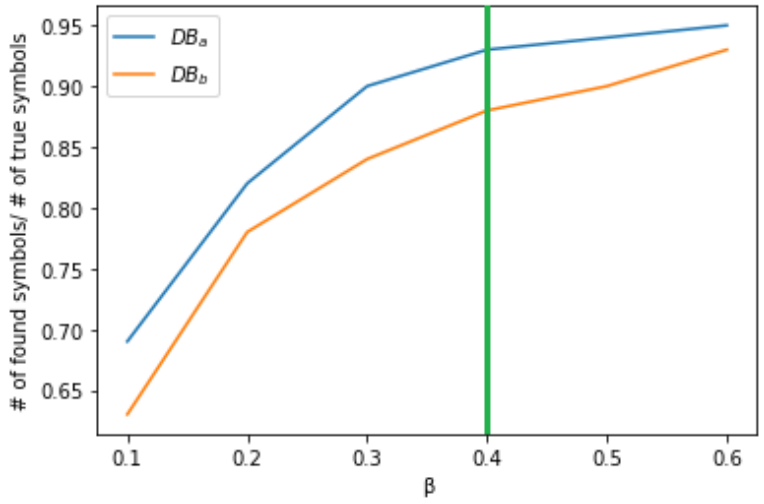
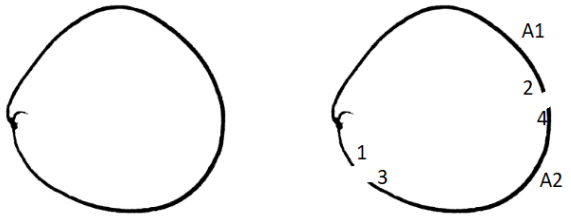


Figure 5.14: The result of experiments performed over DB$_a$ and DB$_b$ to get an optimal value of β coefficient.

FA in Figure 5.15. During segmentation, this state shape may be mistakenly divided into two different symbols, $A1$ and $A2$ (see Figure 5.15(b)). To fix this type of error, we introduced Algorithm 5.3. The role of this algorithm is to produce the original symbol A by joining shapes $A1$ and $A2$, which were divided during segmentation. The idea presented here is built on the fact that more often than not, the shape will break towards its endpoint. If two incomplete shapes with endpoints close to each other are found among the segmented symbols, then the algorithm joins them to form a single shape. For example, in Figure 5.15(b), points 1 and 2 in symbol $A1$ are close enough to points 3 and 4 in symbol $A2$ to form a state symbol, in this case, the algorithm joins $A1$ and $A2$ to form symbol A.



(a) A state symbol A

(b) A state symbol divided into two ($A1$ and $A2$) during segmentation

Figure 5.15: Post-segmentation

Algorithm 5.3 searches through the whole set of segmented symbols SS of the input image for any wrongly segmented symbol (line 2). If a symbol $A1$ in SS is found to be incomplete, then the algorithm checks for the closest symbol to $A1$ that has its endpoints close to the point where $A1$ is divided. A symbol endpoint is said to be

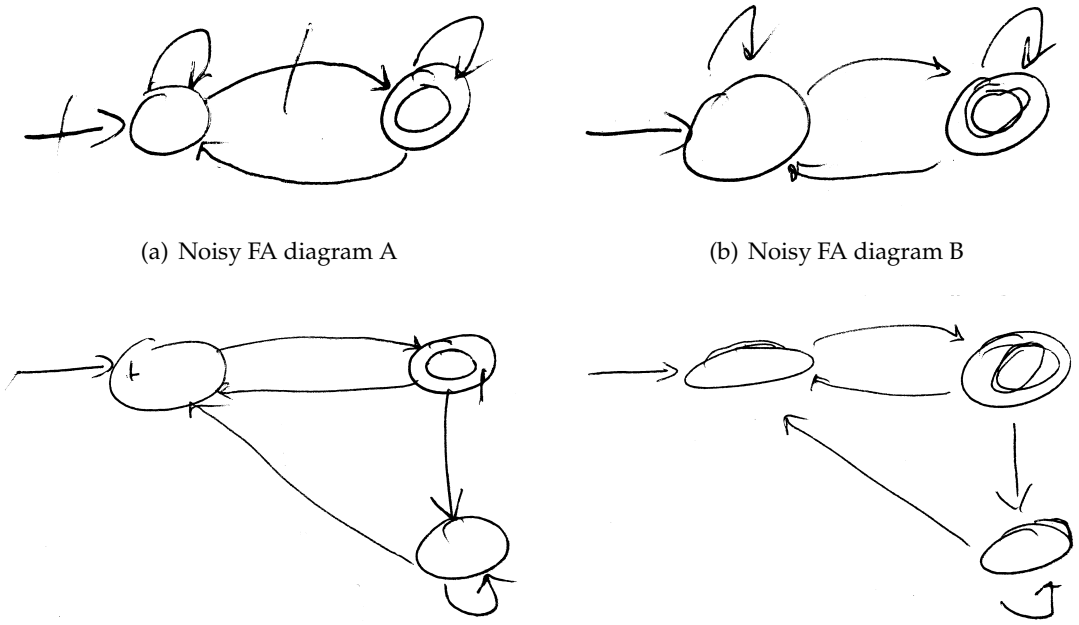
Algorithm 5.3: Post-segmentation

Data: Segmented symbols SS
Result: Complete symbol segmentation
1 initialization;
2 check if there is an incomplete symbol in SS ▷ `If` A `is an incomplete shape in` SS
3 **if** A *is found* **then**
4 identify $A1$, $A2$ ▷ `shapes that will form` A
5 determine if the endpoints of shapes $A1$, $A2$ are close enough to form A ▷ `we used` DT `explained earlier in this section to check the proximity of these shapes`
6 **if** *true* **then**
7 join $A1$, $A2$ to form A
8 **else**
9 discard A
10 **else**
11 find another incomplete symbol in SS

close to another symbol endpoint if the distance between them is smaller than a predefined threshold that could be experimentally pre-determined. Once another symbol that fulfils this requirement is found, i.e. symbol $A2$ in Figure 5.15(b), then the algorithm joins these two symbols together to form the original symbol. To check if $A1$ and $A2$ are truly the components of A, we check if the endpoints of the two symbols are close enough to form A. If the endpoints of these two symbols are close enough, we add these symbols together to form A (lines 5, 6 and 7). If the endpoints are not close enough, then we leave the symbols and search for any other wrongly segmented symbol in SS. As stated earlier, the purpose of this algorithm is to improve the accuracy of the segmentation phase and the entire recognition system. The post-segmentation algorithm performance was good enough as it increased the result of the segmentation phase with 7.3% accuracy.

In this phase, we have introduced a new segmentation method for hand-drawn FA images using bounding boxes and FA component proximity. The algorithm produced a high true positive. Most especially, the segmentation points in circles were correctly detected in most cases. A high false positive rate was noticed in the detection of arrows in DB_b. The primary reason for this is that some arrowheads were drawn overlapping other components. This is a common occurrence in DB_b. Also, the total numbers of false positives identifications obtained in the experiments performed on both datasets (DB_a and DB_b) were similar, despite the fact that DB_a contains drawings that were drawn carefully and DB_b contains drawings that were not drawn carefully. To test the robustness of our method, we chose 30 different FA diagrams from our datasets as test images. We added noise to these diagrams. Figure 5.16 presents some of these 30

diagrams. In Figure 5.16(a), noise was added to the start arrow and the arrow from the start state to the final state. The inner circle of the final state in Figure 5.16(b) was retraced and most of the figure arrows are well isolated from the other components in the figure. In Figure 5.16(c), noise was added to the beginning of the start arrow shaft and also added to the circle of the start state and the final state. Figure 5.16(d) condition is similar to that of Figure 5.16(b) explained above.



(a) Noisy FA diagram A

(b) Noisy FA diagram B

(c) Noisy FA diagram C

(d) Noisy FA diagram D

Figure 5.16: Noisy FA diagrams

Another reason for adding noise to the test diagrams was to make the problem more challenging. In Figure 5.17, NI denotes noisy images. The figure shows that the performance of our method decreases when the diagram becomes more complex and when more noise is present in the image. This is acceptable because it will also be confusing for human with no FA understanding to correctly segment and recognise the final states in Figure 5.16(b) and 5.16(d). Some people might say this final states contains three concentric circles. Another reason for the decrease is that we used a strict approach to evaluate our method. As stated earlier, a diagram can only be said to be correctly segmented if it only contains all the strokes that make it up as they are in the ground truth. This approach makes it more challenging when evaluating segmented diagrams with complex noise. Despite the robustness of our method, when multiple stroke retracing occurs, segmenting such will substantially increase the difficulty of our approach. Multiple tracing is still an open problem in image segmentation generally Li et al. [2018]; Mahdavi et al. [2019].

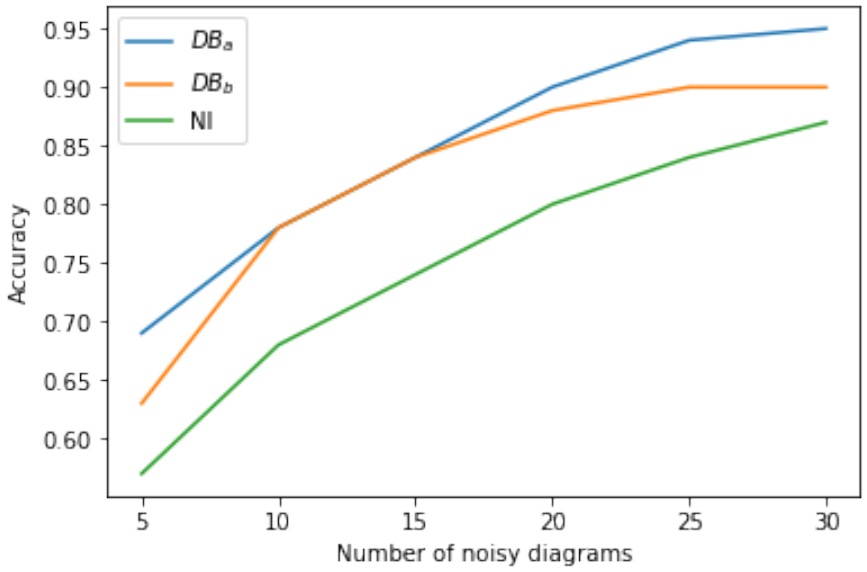


Figure 5.17: Performance comparison of our datasets.

During the evaluation, the segmentation algorithm was analysed on a PC (Debian GNU, stretch, 16GB RAM, 2.9GHz). This system configuration is a typical machine which our recogniser can operate on. The use of this computer during our experiment shows that the recogniser allows real-time performance on average computers. A minimum of 161 milliseconds is required to process diagrams from DB_a, 168 milliseconds is required for diagrams from DB_b, and 252 milliseconds is required for NI. This speed shows that the method has the ability to perform real-time segmentation as users draw their diagrams. At the end of this phase, we have a simplified representation of the input FA diagram that can be analysed and classified easily. Each pixel in the input image has been given a label such that all pixels with the same label have certain features such as geometric and visual features in common.

5.5 Arrow Detection and Recognition

Most research that has been done on the recognition of connected diagrams such as flowcharts, UML diagrams and FAs has treated the arrow as a symbol that exists between two other symbols [Bresler et al., 2015]. This approach has been effective in recognising arrows in flowcharts, graph diagrams and UML diagrams. Using this approach in our research will generate a good result when recognising arrows that exist between two circles (i.e. two circles connector), but will be less efficient when recognising the start arrow and the loopback arrow in an FA diagram because the start arrow

does not exist between two other symbols, likewise the loopback arrow too. See Figure 5.18 for these types of arrows. To recognise arrows in FA diagrams effectively irrespective of the type of arrow, we introduced a new approach here. This new approach uses the structure of an arrow and ignores its position during recognition. Using the structure of arrows for recognition will allow different shapes of arrows to be detected and recognised irrespective of their position in the diagram.

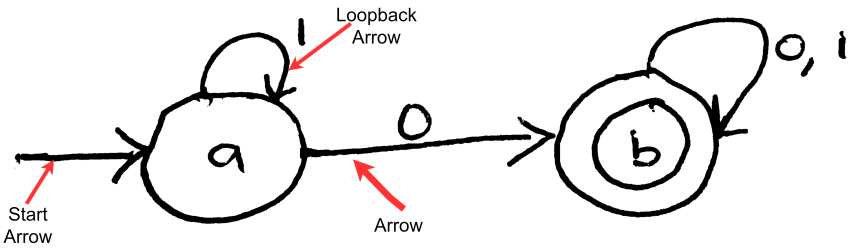
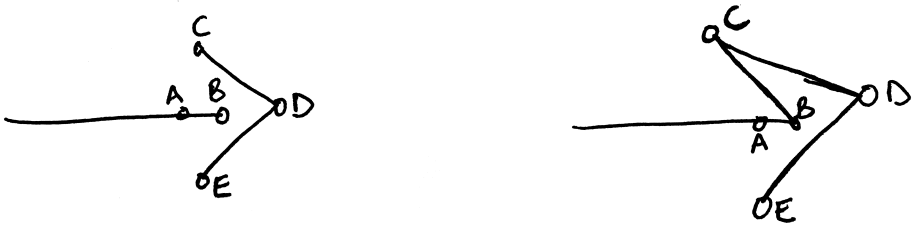


Figure 5.18: Types of arrows in an FA diagram

During this phase, we defined three types of arrows for FA diagrams, namely, the start arrow, an arrow (connecting two circles) and the loopback arrow (see Figure 5.18). The algorithm we used here treats all the types of arrows the same. To increase the accuracy of the arrow recogniser, we made an assumption on how people tend to draw arrows. We assumed that most users draw an arrow from the shaft to the head. Also, most arrows are drawn as two consecutive pen strokes and only in a few cases as a single pen stroke. These two points were taken into consideration when designing the arrow recogniser. Similar to what was done in Kara and Stahovich [2007], we identified five key points on an arrow (see Figure 5.19). These points are A, B, C, D and E in Figure 5.19. They are used to determine the arrow's shaft and head. The approach described in Kara and Stahovich [2007] focuses on recognising single stroke arrows only (see Figure 5.19(b)). Based on our assumptions, we extended their approach to recognising a two stroke arrow (see Figure 5.19(a)).



(a) Two consecutive strokes of an arrow

(b) Single pen stroke of an arrow

Figure 5.19: Drawing of an arrow

In Figure 5.19(a), B is the end-point of the first stroke, C is the starting point of the second stroke, followed by points D and E. In Figure 5.19(b), we choose any point

close to point B on the arrow shaft as the first point of interest. Point A is picked as a point of interest on the stroke and is a short distance from B, points B, C and D are the corners in the stroke, and E is the last point in the stroke.



Figure 5.20: Example of varieties of an arrow shaft

During this phase, the input to the algorithm (i.e. Algorithm 5.4) is obtained from the graphics strokes. From the group of graphics strokes, we are interested in those segments assigned an arrow label. To detect and recognise the strokes with an arrow label, we focus more on the head of the arrow and less on the arrow shaft; this will allow for a variety of shapes for the arrow shaft or drawing style as illustrated in Figure 5.20. Before the proper recognition, we have to determine if the pen stroke is truly an arrow. To do this, a geometric test was conducted [Kara and Stahovich, 2007]. Four angles $\widehat{BCD}$, $\widehat{CDE}$, $\widehat{ABC}$ and $\widehat{ABE}$ are required for this test. These angles must all be less than $90°$ as shown in Figure 5.21 and the length of the line segments $\overline{CD}$ and $\overline{ED}$ must be less than 20% of the total length of the stroke.

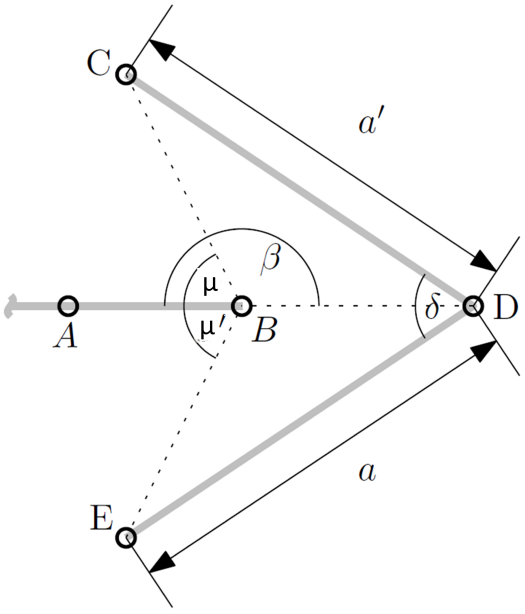


Figure 5.21: Features used for arrow recognition

To extend this approach to recognise two-stroke arrows, we check if points B, C, D and E are at least part of a single stroke, i.e. a single pen stroke arrow. These points are found in the group of strokes tagged with an arrow label during the segmentation

Algorithm 5.4: Processing arrow strokes

Data: Hand-drawn strokes
Result: Processed arrow strokes
if *points C, D and E are part of a single stroke and close to point B in the figures in Figure 5.19* **then**
    rotate all points so that line AB is horizontal, and A is on the left of B
    scale all points so that their bounding box fits into the square $[-0.5, 0.5] \times [-0.5, 0.5]$
    extract the numerical features using the angles β, δ, μ and μ', and the length a and a' (see Figure 5.21)
else
    find another arrow stroke

phase. To recognise a two-stroke arrow from this approach, we aim at splitting such a stroke into two, i.e. shaft and head of the arrow using the steps below, aiming at finding an arrowhead;

(i) Check for a single pen stroke (Figure 5.22) that exists between two circles.

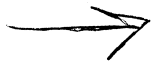


Figure 5.22: Single-stroke arrow

(ii) Mark this stroke as an arrow that could possibly have been made with a single stroke of the pen.

(iii) Split the stroke towards the end (the part closest to the head) of the shaft into bits (i.e. sub-strokes) as shown in Figure 5.23. The result of this will be a shaft separated from the head.



Figure 5.23: Divided arrow stroke i.e. sub-strokes

(iv) Divide the sub-strokes in (iii) above into two classes. See Figure 5.24 for these different classes –

- Class one sub-strokes are added to the split shaft. This is to allow the shaft to reach the next symbol to it.

- Class two strokes are grouped together as a set of strokes that are possibly forming the arrowhead.

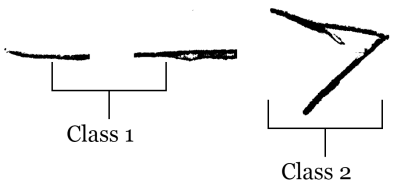


Figure 5.24: Sub-strokes classes

As explained earlier, most of the arrow symbols in our datasets are drawn as two consecutive pen strokes. The arrow recogniser we introduced was tested on arrows from our datasets. In DB_a, we have 27,894 strokes that form 13,932 arrows, and DB_b contains 5,203 strokes that form 2598 arrows. The arrow classifier achieved an accuracy of 94.0 and 85.0 on DB_a and DB_b respectively at the symbol level. The result of the evaluation shows that our approach is robust and handles writing irregularities effectively. Most cases of misclassification were reported in DB_b. This is because most arrows in DB_b were not drawn carefully. Irregularities like arrow shaft isolated from arrowhead and arrow shaft not having an arrowhead were some of the occurrences in this dataset. At the end of this phase, the arrow symbols of an FA diagram have been detected and recognised among other components of an FA diagram. Once the arrow symbol has been recognised, we are left with the state symbols.

5.6 Shape Detection and Recognition

After detecting and recognising the arrow symbols in the graphics component, we are left with state symbols. To classify a symbol as belonging to the class state, we used support vector machines (SVMs) to train the datasets in DB_a and DB_b. At this stage, the data are states or accepting states. An SVM (explained in section 2.8.1) is a powerful discriminative classifier that has been widely used in many classification [Ghaddar and Naoum-Sawaya, 2018] and recognition [Kanisha et al., 2018; De Marsico et al., 2019] problems with good results. It can be formally defined as a separating hyperplane, i.e. in supervised learning, if given a labelled datum, the machine outputs an optimal hyperplane. In this research, we used the visual and geometric features proposed in Zhang and Lu [2002] as a feature descriptor. The descriptor is based on moment invariants which are derived from moments of shapes and are invariant to 2D geometric

transformations of shapes. In this descriptor, a feature vector is made up of seven moment invariants: $f = (\Phi_1, \Phi_2, ..., \Phi_7)$ as described in Section 2.5.4. These are used to index each shape in the databases. The main reason for using this descriptor is that it is computationally effective and compact to shape representation. These features are used as input to the SVM classifier. The datasets from DB_a and DB_b were used in training the classifier. If the input cluster of strokes is annotated as a state class in the databases, it is then marked as a symbol to be classified. If it is labelled as a no match (i.e. unused strokes of arrows or text), it is denoted as a negative example.

The system detects these stroke classes by extensively scanning the whole dataset for state-like patterns at many possible scales. This is done by dividing the original diagram into overlapping sub-diagrams and classifying them using an SVM to determine the right class (i.e. state or final state). The following steps were followed:

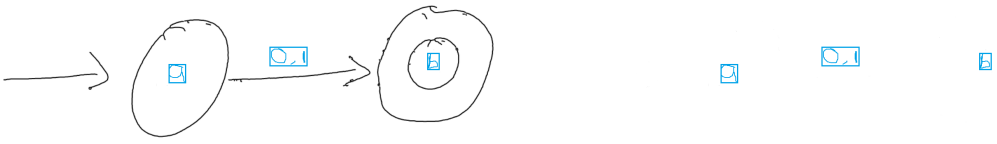
- A dataset of state patterns (start state and final state) using their geometric and visual features is presented and each pattern is assigned to a class, i.e. positive and negative class. We use these to train the SVM with a second degree polynomial as a kernel function.
- To further pre-process the image, i.e. to generate a good classification, histogram equalization is done to improve the contrast in the patterns.
- Once a decision has been taken during the training process, images that are misclassified during the training process (i.e. a state taken for a final state) are not removed but stored. They can be used during subsequent training phases as negative examples.
- After the SVM training, we used the classifier as a run-time system. After detecting patterns and then assigning them to a class, the classifier gives different colours to the classes. It uses green for a final state and blue for a non-final state.

To test the implemented classifier, we used DB_a and DB_b. Multiple comparisons based on false positive ratio which is the probability of falsely rejecting the null hypothesis for a particular test was performed. To give a true meaning to the number of false positives obtained, it is necessary to report that DB_a contains 3908 pen strokes of states and 3186 pen strokes of the final state and DB_b contains 597 pen strokes of state and 591 pen strokes of final state patterns. The detection and test accuracy on DB_a was 96.2% and 90.4% on DB_b at stroke level.

5.7 Text Detection and Recognition

The input for this phase is the bounding box of the text layer (see Figure 5.25(b)) extracted from the whole diagram during the text and graphics separation phase. We used Tesseract OCR (Optical Character Recognition) [Smith, 2007] to process the contents of the bounding boxes. Tesseract[2] is an open-source text recogniser engine which was designed as a prototype in the University of Nevada in Las Vegas (UNLV) Fourth Annual Test of OCR Accuracy. We chose Tesseract among other OCRs because it is generally referred to as the most accurate open source OCR engine available to recognise text from images and scanned documents.

The major challenge with this OCR engine is that it performs better when the foreground text is well separated from the background. In practice, it can be extremely challenging to guarantee this type of segmentation. However, since the FAs are basically drawn with black ink on white paper in this research, this type of segmentation is obtainable. To do this, we trained the image classifier for our domain of interest. Training classifiers for domain-specific image classification is a common approach now used by researchers. During our experiment, since Tesseract was mainly developed for words and sentences, the page segmentation mode was set to a single character because in most cases FA labels are single characters and not words. We noticed that it is important that text characters do not appear pixelated after segmentation using the OCR. Instead, their resolution needs to be high. Tesseract encountered difficulty in recognising characters that are pixelated in DB_a and DB_b.



(a) All the text strokes are classified in blue bounding boxes

(b) All the text stroke bounding boxes after removing the graphical component

Figure 5.25: Text detection and recognition in an FA diagram

The accuracy of the text recogniser was tested by comparing its output with the values of the input text and we got 90%. We chose 100 images from the data. The OCR was able to accurately recognise 90 out of 100 text characters presented to it. Of the ten unrecognised characters, five are text characters with subscripts. The failure of the OCR to recognise this can be attributed to the fact that it was primarily developed to recognise alphabet letters and numbers and not mathematical expressions. Another error that can be encountered with the OCR is that if a text character is badly written, the

[2] https://github.com/tesseract-ocr/tesseract

system may consider it as an unused graphics stroke instead of a text character. Bad pixelation can also be a reason for system failure.

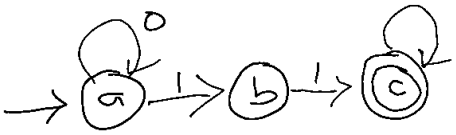
5.8 Final Recognition Result

After the input diagrams have been processed in all phases, we now have a final output. The output of the detected and recognised components of the FA diagrams are illustrated with a few examples in Figure 5.26. The final output of the entire recognition process is illustrated in Appendix A. The recognition system has high accuracy. However, some diagrams were not recognised correctly. The system failure is explained in detail in Section 6.6. Consider the FAs in Figure 5.26, all components of an FA diagram have been detected and recognised accordingly. Each component of the diagram with its corresponding bounding box was labelled with a unique colour start arrow - red, state - blue, arrow - yellow, text - purple, final state - green.

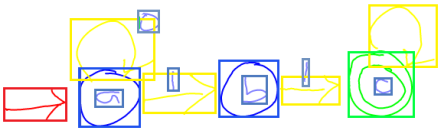
5.9 FA Transition Table

The input of this stage is the whole recognised FA diagram with a focus on making the recognition system understand the relationships that exist between the different components of the diagram. During the process of recognising the diagram, the order and layout of the recognised symbols are maintained i.e. the symbols are in the same position as they are in the ground truth diagram. For example, in Figure 5.26(d), the start arrow for the final recognition output will be at the top while it will be at the left in Figure 5.26(f). Since the layout of the diagram is not lost during recognition, the relationship that exists between the components of the recognised FA diagram (i.e. an arrow or arc from state p in Figure 5.27 to state q has the label 1) can be identified easily. This process is explained later in this section. Hence, generating transition tables for the recognised diagrams is possible.

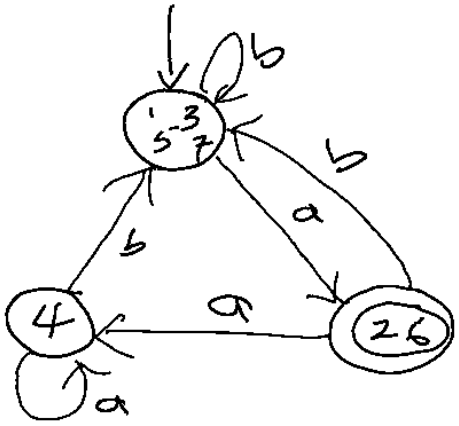
A transition table is a tabular representation of the transition function (δ) that takes two arguments and returns a state. The row in the table corresponds to the state the machine can be in. The column contains the state in which the automaton will be on the input presented by the column. For example, the entry for one row corresponding to a state p and the column corresponding to input 1 will be $\delta(p, 1)$. The transition table of Figure 5.27 is presented in Table 5.1.



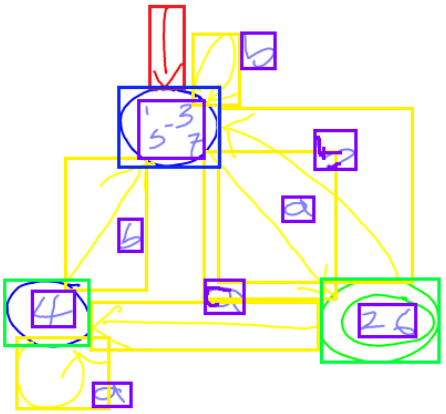
(a) Original binarised image



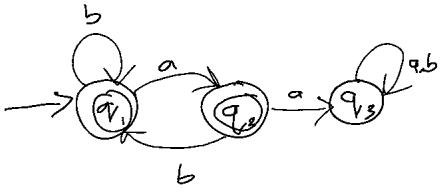
(b) Corresponding output



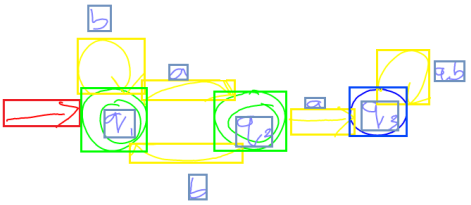
(c) Original binarised image



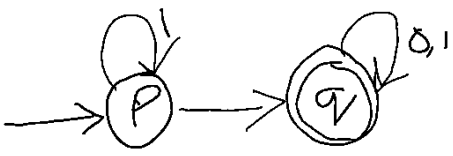
(d) Corresponding output



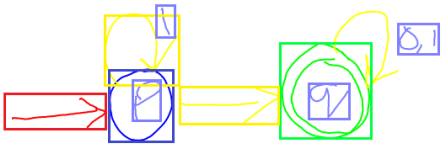
(e) Original binarised image



(f) Corresponding output



(g) Original binarised image



(h) Corresponding output

Figure 5.26: Output of all recognised components of the input FA diagrams

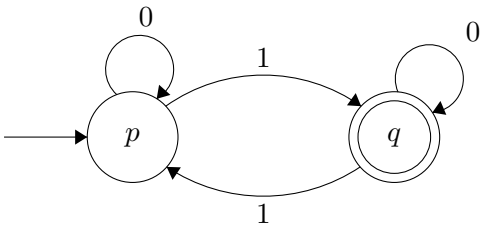


Figure 5.27: An FA diagram

Table 5.1: Transition table representation of Figure 5.27

	0	1
p	p	q
q	q	p

Formally, the automaton is $M = (\{p, q\}, \{0, 1\}, \delta, p, \{q\})$ where –

- $\delta(p, 0) = p$
- $\delta(p, 1) = q$
- $\delta(q, 0) = q$
- $\delta(q, 1) = p$

To generate the transition tables of the recognised FAs, we use the recognised text labels of Section 5.7. The approach used here starts by taking note of the recognised text labels inside the classified state symbols as described in Section 5.6. Since the state symbols have been classified into their respective classes i.e. start state, normal state and accepting state in Section 5.6, the approach can efficiently identify the type of state it is in and the label inside the current state can be noted. The approach can identify a symbol classified as an arrow using the knowledge from arrows recognition discussed in Section 5.5. When a symbol is classified as an arrow, the label on the arrow is also noted. Once all the labels have been noted, we focus on identifying the relationship that exists between the different components of the recognised FA diagrams. The algorithm starts by creating a list to store identified labels. To identify the relationship, the algorithm starts by taking the start state from the group of recognised components and add its label (i.e. p in Figure 5.27) to the created list. It then checks if the current state (i.e. the start state) have loopback arrows. If loopback arrows exist, then the algorithm visits each of the arrows and note the label on the arrows (i.e. 0 in Figure 5.27), else it checks for the next transitions from the current state. The algorithm returns the noted label and stops if no arrow transition exists from its current state to another state. However, if an arrow exists, it then takes note of the label on the arrow and note the

transition between the previous state and the current state (i.e. state q in Figure 5.27) which is an accepting state in Figure 5.27 but can be any of normal or accepting state generally. Before the algorithm accepts the input and stops, it checks its current state for other transitions. If it detects an arrow transition from the current state to another state (i.e. label 1 from state q to state p), it will transition back to state p and take note of the label before adding it to the list. This process is repeated until all the transitions label have been noted in the transition table.

Using the approach explained above, we generated the transition tables for the recognised FA diagrams. The accuracy of the transition table algorithm is 86.1% and 84.4% on DB_a and DB_b respectively. The algorithm still cannot identify accurately the whole relationship that exists between FA components and may return a few errors when generating tables for FAs with many states and transitions.

5.10 Summary

The major phases of the recognition system were discussed in this chapter. The approach used in labelling the strokes in the benchmark database was explained. To reduce the complexity of the system and increase the system speed, we separated the graphic components of the FA diagram from the textual components. The system achieved higher and better accuracy on database Db_a (97.2%) compared to DB_b (85.4%) during the graphics and text separation. These results can be attributed to the fact that data in DB_a are well drawn compared to those in DB_b. The components of the separated graphical components (states/circles and transitions/arrows) were segmented.

After segmentation, we focus on detecting and recognising these different components. The segmentation algorithm used here explored the structure of an FA diagram (i.e. different components are connected together to form a single diagram) to segment it into its different components. Due to the complexity and challenging nature of offline handwriting segmentation, a post-segmentation algorithm was introduced. This algorithm aims to correct wrongly segmented symbols. This chapter showed why the bounding box segmentation approach is better than stroke level segmentation for diagrams like FAs. During the symbol recognition phase, three different tasks were performed, namely: arrow detection, shape detection and text detection. During the text detection phase, Tesseract OCR was leveraged on to achieve the aim of the phase. At the end of this phase, we have successfully detected and recognised all the components of an FA diagram. The corresponding transition tables of the recognised FA diagrams were generated.

The next chapter reports on the evaluation of the recognition system. The major reason for evaluating is to compare the output of the recognition system with the ground truth data and to calculate the accuracy of the system. Also, the parameters used during the evaluation and system analysis process will be discussed. The recognition system failed to recognise some components of the data presented to it; this is also presented in Chapter 6.

Chapter 6

Experimental Results and Discussion

6.1 Introduction

A detailed description of the recognition system, from the pre-processing phase through to the final outputs has been given mainly in the previous chapter. We have explained with illustrations the input of each of the recognition phases and the expected output. We have stated the system accuracy at each phase. The approach to generate transition tables for the recognised FAs was also explained in the previous chapter.

In this chapter, we describe the dataset used for this study. We explain the evaluation process and techniques used to evaluate the FA recognition system. All experiments and the evaluation done in this chapter were performed on the created databases which are introduced in the next section. One reason for evaluating the recognition system is to objectively measure the performance and accuracy of the system. Another reason is to understand the behaviour of the recognition algorithm in recognising various classes of diagrams. According to Heath et al. [1997], evaluating the performance of a recognition algorithm depends on the following factors:

- the nature and type of datasets used in measuring the algorithm's performance,
- the system or algorithm itself,
- the approach used in evaluating the system, and
- the algorithm parameters used in the evaluation.

6.2 Datasets

As explained in Section 2.7, there is still a limited dataset for offline recognition of hand-drawn diagrams. At the time of writing this thesis, there is no existing dataset for offline FA diagrams to the best of our knowledge. We have collected our data. This has been explained in Section 3.2. Initially, we created a dataset of hand-drawn FA images that were drawn mostly by students who have taken and passed a course in formal language and automata theory. These images were store in a database called DB_a. During data collection, participants were all informed that these diagrams will be used for research purposes. To this end, most of the images in this database were drawn with little or no error. Hence, we called this dataset synthetic. To investigate how accurately our system can recognise less carefully drawn FA diagrams, we created another database of FA diagrams called DB_b. DB_b contains diagrams taken from student past examination papers. Most of these diagrams are not well drawn and contain irregularities such as FA diagrams with overlapping strokes, circles drawn as ellipses, and isolated components. Having these two databases allows us to investigate how the recognition system can perform on more difficult images. A detailed analysis of the datasets and their mode of acquisition was explained in Section 3.2.

6.3 Database

We have created two offline databases for hand-drawn FA images. The first database DB_a contains the synthetic data. The second database DB_b contains the raw data. The major reason for having these two databases is to investigate how quickly our system can adapt to learning to recognise less carefully drawn diagrams. What each of these data looks like has been explained in the previous section.

We used a Samsung s8 mobile phone camera to capture all the diagrams for both databases. All images contain a similar level of noise since they are all drawn on the same type of paper and captured with the same device.

Image Annotation Image annotation is the task of labeling the components of a diagram. This process is usually done with text and with some additional details such as the print conditions and details of the image, script or language, and layout information. In this research, labeling of an FA diagram component is done in two stages i.e. the segmentation and classification stages.

To do this, we have developed a tool (see Appendix C) for the annotation of these diagrams. The developed tool was designed for bounding boxes since that is

enough for labeling FA diagrams, and does not support annotation of rotated rectangle regions, and 3D. This tool is capable of performing various labeling operations on connected diagrams.

During the annotation process, FA diagram components were divided into classes i.e. start arrow, circle, final state, text and arrow. The mode of operation of each of the annotation stages is listed below:

(i) The segmentation stage starts by the user adding an FA diagram to the annotation interface presented in Appendix C.1. Then they select the type of segmentation tool to be used. The segmentation tools are rectangle and lasso. Since the focus is on the bounding box tool, the rectangle segmentation tool was mostly used during the labelling instead of the lasso tool. To create a segmentation, the user selects the class to be segmented from the image and then draw a box around a representative of a class of interest (see Appendix C.1). For example, if the user wants to segment the start arrow, they will draw a rectangular bounding box around the start arrow to segment it from the diagram.

(ii) The classification stage involves adding the segmented component to its class. For example, if a user segments a start arrow, they will be prompted to add it to any of the predefined classes. It is important a component is added to the right class, as this will give a high recognition accuracy and avoid component misclassification at a later stage. These are shown on the right pane of Appendix C.1.

Appendix C.2 shows the interface and the segmentation and classification tools. Each annotated diagram is saved in an XML file format separately in its corresponding database.

6.4 Experiment

The evaluation of the offline hand-drawn FA diagram recognition system was performed on the benchmark datasets i.e. DB_a and DB_b. The DB_a dataset contains over 15,000 symbols of circles, arrows and text while DB_b contains over 3,000 symbols. An overview of the datasets is presented in Table 6.1. In Table 6.1,

- "Writers" are the number of participants,
- "Patterns" means the number of different FA structures that were drawn,

- "Diagrams" means the total number of FAs drawn by all the writers,
- "Strokes" are the number of strokes in each dataset and
- "Symbols" are the number of symbols i.e. circles and arrows.

Table 6.1: FA datasets

Database	Writers	Patterns	Diagrams	Strokes	Symbols
DB_a	50	10	500	40,855	19,568
DB_b	100	2	200	11,102	3,801

Table 6.2 shows a detailed overview of the datasets in Table 6.1 after separated into training and test datasets. We have used sufficient training dataset in this research. We used 500 images for training because there are only a few possible symbols and patterns when drawing an FA diagram. Also, with 150 writers for both datasets, we have enough variation in the drawing style. Hence, the used training set will be sufficient to recognise most FA diagrams accurately.

Table 6.2: FA database overview

Database		Writers	Patterns	Diagrams	Strokes	Symbols
DB_a	Training set	50	10	350	26,426	12,862
	Test set	50	10	150	13,621	6,020
DB_b	Training set	100	2	150	8,320	2,852
	Test set	50	2	50	2,782	949

It is important to state that the drawing order of the diagrams was not considered since our focus is on offline recognition. The drawing order can only be known if the recognition system is an online recognition system. During the experiment, the recognition system was analysed on a PC (Debian GNU, stretch, 16GB RAM, 2.9GHz). This system configuration is a typical machine which our recogniser can operate on. The use of this computer during our experiment shows that the recogniser allows real-time performance on average computers. The minimal, maximal and average running time in milliseconds used by each of the recognition phases during the entire recognition is presented in Table 6.3. The values show that the whole process is relatively fast.

6.5 System Analysis

The evaluation of the system was done on two levels, namely, the stroke labeling level (α) and the symbol segmentation and recognition level (β). The purpose of this evaluation is to check if the output contains the same number of strokes and symbols as the ground truth, and also to check each stroke label against its ground truth in the

Table 6.3: Running time in milliseconds taken by each phase.

Phases	DB_a			DB_b		
	min	max	avg	min	max	avg
Text and graphics separation	20	195	80	20	230	84
Stroke extraction	480	10,420	2,370	900	19,821	6,364
Segmentation	161	20,089	2,534	168	24,521	2,571
Classification	408	2,521	1,542	382	2,947	1,754
Text	24	201	80	25	274	90

database. Since each stroke has a label, the output of this level is compared with the ground truth in the database. On the stroke labeling level, each stroke is given a label of the symbol it belongs to. To check if this label exists in the ground truth, the nearest annotated stroke with maximal overlap is found. On the symbol segmentation and recognition level, an annotated symbol is said to be segmented and recognised correctly if there is a match between detected symbols. If two symbols labels match and their bounding boxes overlap at least 85%, then these symbols are said to match. During the recognition phase, some symbol strokes may be invalid or some diagrams may contain invalid symbols that do not exist in the ground truth. To measure the accuracy of strokes in such a scenario, we used recall.

Table 6.4: Recall of stroke labeling accuracy and symbol recognition

Database	α	β
DB_a	97.1	92.3
DB_b	92.0	85.7

The recognition system achieved high precision as shown in Table 6.5. Here, the values for each symbol class represent its recognition accuracy in percentage at the two evaluation levels. Also, the average of each recognition type for both databases is computed as the macro-accuracy of all the symbol classes which is the sum of the recognition accuracy of all the symbol classes divided by the number of symbols. In this table, DB_a performed better when compared to DB_b for stroke and symbol recognition with an accuracy of 95.4% and 94.0% respectively. This is simply because of the nature of the data in each of the databases which have been explained in previous sections. When analysing the performance at the symbol class level, text seems to be the symbol with the lowest accuracy. This can be due to the fact that most FA texts in the datasets contain subscripts which are classified as mathematical expressions and the OCR we used is primarily developed to recognise alphabet letters and numbers and not mathematical expressions. With respect to the state class, if the user writes a label

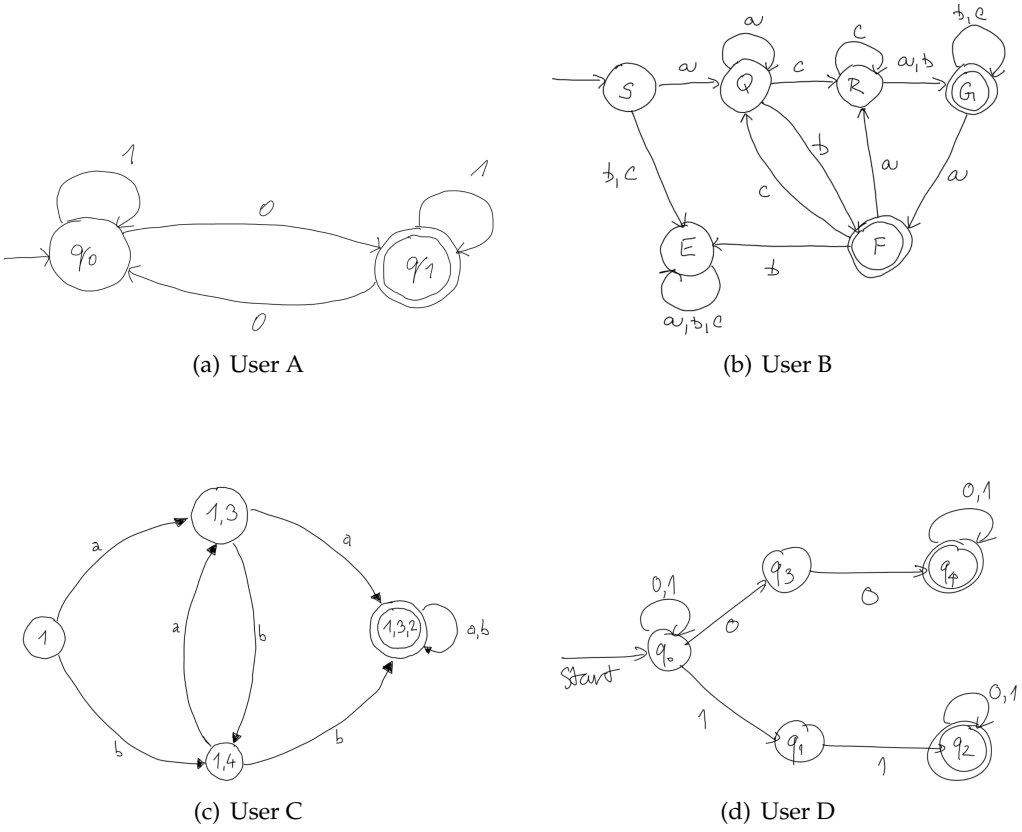
such as *C* or *O* carelessly inside the class (e.g., the label is touching the symbol), the evaluation may fail here as the system may classify that state as a final state.

Table 6.5: Recognition system accuracy at stroke and symbol level

Symbol Class	Stroke Recognition		Symbol Recognition	
	DB_a	DB_b	DB_a	DB_b
State	96.2	90.4	94.5	87.9
Final state	96.8	91.0	95.7	87.6
Text	90.0	90.0	90.0	90.0
Start arrow	97.4	89.8	96.0	87.6
Arrow	96.6	87.6	94.0	85.0
Average	95.4	89.8	94.0	87.6

As explained earlier, there was no offline database for FA diagrams in literature. To do a comparison of the approach presented in this work with other work, the only existing online database of FA diagrams, presented in Bresler et al. [2015], was used. The database contains 12 FA diagram patterns drawn by 25 different people. All 300 FA diagrams in the database were annotated and split into 132 training sets, 84 validation sets and 84 test set. The comparison was done at stroke level and symbol level as explained earlier in this chapter. We represent the online database as DB_{on} in this chapter. The data created and stored in DB_{on} were collected with a PC tablet and stylus so these data are noise-free and do not need to be thresholded and thinned. Also, stroke extraction may not be necessary for the datasets. Figure 6.1 illustrates some FA diagrams taken from DB_{on}.

Table 6.6 shows the comparison of our work with Bresler et al. [2015]. The average runtime on DB_a is 0.8 seconds, DB_b is 13.0 seconds, and DB_{on} is 0.92 seconds. From the symbol recognition result in this table, it is to be noted that the recognition accuracies on DB_{on} with regards to states and final states are higher than on our datasets. This could be because the shape recognition approach we used can easily be affected by sample bias; the number of states and final states shapes is less than arrows and text. Separating these shapes from others will significantly improve the recognition accuracy of these shapes class i.e. states and final states. Also, it is worth pointing out that our approach in recognising the arrow class has given a better recognition result on dataset DB_a when compared to the results obtained for DB_{on} and DB_b. This is simply because the approach described in Bresler et al. [2015] is based on human prior knowledge that sees an arrow as a symbol that connects two other symbols. Hence, making recognition of different shapes of arrow difficult for the algorithm proposed in Bresler et al. [2015]. The drawbacks of recognising arrows as two symbols connector and why we did not use the approach were explained in Section 5.5. The results on DB_b is lower than the two other datasets because the data in DB_b contain several irregularities that affected



(a) User A (b) User B

(c) User C (d) User D

Figure 6.1: Sample FA diagrams from DB_{on}

the stoke and symbol recognition process. These irregularities were highlighted in Section 3.2.2. The comparison presented in Table 6.6 compares our datasets and DB_{on} at stroke and symbol level.

Table 6.6: Comparison of our datasets and DB_{on}

Symbol Class	Stroke Recognition			Symbol Recognition		
	DB_{on}	DB_a	DB_b	DB_{on}	DB_a	DB_b
State	96.1	96.2	90.4	95.0	94.5	87.9
Final State	94.4	96.8	91.0	97.2	95.7	87.6
Text	99.0	90.0	90.0	99.0	90.0	90.0
Start Arrow	96.2	97.4	89.8	94.1	96.0	87.6
Arrow	92.2	96.6	87.6	93.1	94.0	93.1

Table 6.7 compares the total running time (processing and recognition time) of our databases with DB_{on} when analysed using the recogniser. The result shows that DB_a has a lower average running time than the two other databases. Also, the recogniser performs better when recognising diagrams from DB_{on} than when recognising diagrams from DB_b as shown in Table 6.7. The stroke extraction and segmentation phases take more time during the recognition process when compared to other phases. It is

possible to optimize both the stroke extraction and the segmentation phases by running them in parallel. Optimization of these two phases will reduce the recognition time. We did not explore this because offline recognition does not need real-time performance. Also, the current average running time is good enough.

Table 6.7: Total running time in seconds

Database	Running time		
	Min	Max	Ave
DB_a	0.31	2.82	0.65
DB_b	0.76	4.16	0.91
DB_{on}	0.21	2.61	0.75

For the transition table generator, we compare the transition table generated with that of the ground truth. Some errors occurred at this phase such as the system cannot read subscripts used in labeling the state symbols, for example, q_0 will be interpreted as $q0$ or qo. Some states labelled as p are interpreted by the recogniser to be 9 in the transition table. However, the arrow label(s) are mostly recognised accurately. This can be due to the fact that arrow label(s) do not have a subscript. The transition table generator achieved an accuracy of 86.1% on DB_a and 84.4% on DB_a. The results of each of the components are presented in Table 6.8.

Table 6.8: Accuracy of each symbol class label

Symbol class	DB_a	DB_b
Start state	84.6	81.7
Final state	82.4	80.2
Transition	91.3	91.3
Average	86.1	84.4

6.6 Misclassification

Notwithstanding the high accuracy of our system, the system failed to recognise some strokes and symbols. It is important to state that the recognition system can fail due to any of the following reasons,

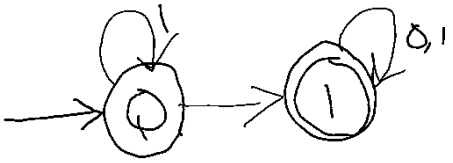
(i) Bad illumination while capturing the image.

(ii) Poor camera quality.

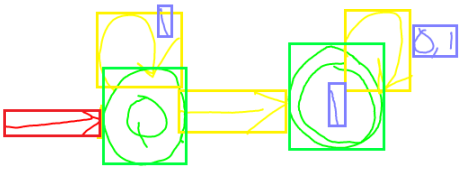
(iii) Badly written text or stroke.

(iv) Improper segmentation.

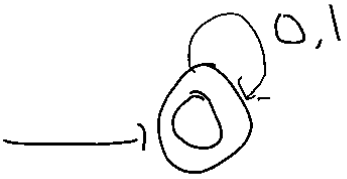
Some examples of these errors are shown in Figure 6.2.



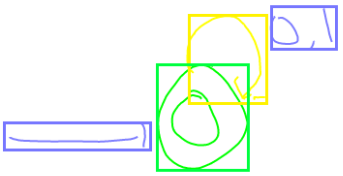
(a) Original input



(b) Label Q is misclassified as a circle, hence the recogniser sees the state as a final state



(c) Original input



(d) Start arrow misclassified as text due to the arrowhead

Figure 6.2: Illustrations of misclassification from our datasets

As explained in Section 5.4, the segmentation phase is the major process that causes incorrect recognition in most image processing tasks. In this work, most of the segmentation errors occurred on data in DB_b and few on DB_a. The main reason why DB_b have more errors during segmentation can be due to the fact that the diagrams were written with no instructions (i.e. writers drew these diagrams without any constraints). Hence, some components in the diagram were retraced by the writers (see Figure 6.3).

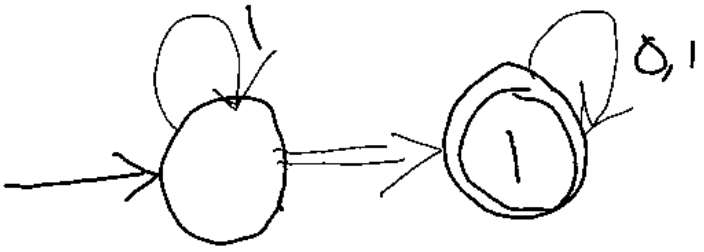


Figure 6.3: Arrow retraced

The classification phase also sometimes fails, hence causing wrong recognition. This happens when the classifier misclassifies a component. For example, in Figure 6.2(d)

the start arrow is classified as text; the likely cause of this is the structure of the arrowhead.

6.7 Findings

During our experiment, we noticed irregularities not only in the data in DB_b, but also for FAs in DB_a despite the mode of their collection. Also, we found out that the bounding box of an FA diagram component and the context available help in symbol class recognition. Different work that has been done in the field of connected hand-drawn diagrams used grammar for their domain of application [Costagliola et al., 2007; Damen and Hogg, 2009; Simistira et al., 2015; Chen et al., 2015b; Roudaki et al., 2016]. This makes using and extending such a system for a new domain (i.e. adopting a grammar-based FA diagram recogniser for a UML diagram) challenging because new rules must be defined for the new domain of application. Each component of our system has been designed to suit its function best. When developing an offline recognition system, it is important to focus the segmentation stage on the domain of interest [Graves and Schmidhuber, 2009; Wu et al., 2015]. This will improve the accuracy of the system and also increase the confidence of the used classifier and the recognition system in general. Also, a training set and test set were created to train and develop the recognition system. The result of the different analyses performed on the system showing the effect of each phase is presented in Table 6.9. We compared it to the baseline used in obtaining the result presented in Table 6.5. The following was measured during the analysis; the stroke labeling (α), the symbol segmentation and recognition (β) and the average recognition time in milliseconds.

We believe that the recognition system has shown promising results and the approach presented in this research can be adjusted with a little effort to recognise other structured diagrams. We have explained the approach used in evaluating the recognition system in this chapter. The next chapter presents the conclusion of this thesis and future work.

Table 6.9: Analysis presenting the effect of each phase on the entire recognition process on our databases

(a) DB_a

	Baseline	Text and graphics separation			Extraction	Segmentation	Classification
		Unbiased	Ground truth	No separation			
α	95.4	95.4	97.2	84.2	92.0	95.1	95.2
β	94.0	94.0	95.8	82.4	91.3	94.6	93.8
Average Time (ms)	715	722	676	941	2370	2534	1542

(b) DB_b

	Baseline	Text and graphics separation			Extraction	Segmentation	Classification
		Unbiased	Ground truth	No separation			
α	89.8	89.8	91.2	81.4	86.5	88.1	89.2
β	87.6	87.6	89.8	79.6	84.3	86.4	85.7
Average Time (ms)	825	874	705	1272	6364	2571	1754

Chapter 7

Conclusion and Future Work

As document mode of acquisition and storage are shifting from the traditional approach to the computerised approach, systems that have the ability to convert a handwritten document into a digital version are needed. Most documents contain text characters and images that are handwritten, typed or printed. This has led to the development of OCR that has had advancement since the 1980s, leading to the development of pattern recognition systems [Mantas, 1986; Stoller, 1973; Hammond and Davis, 2006; Bluche, 2015]. In particular, technologies that have been developed in the last decades now allow people to enter a text by writing directly on a PC tablet or touchscreen. As explained in Chapter 1, sometimes typing text is faster than writing text. However, there is no doubt that it would be more difficult and time-consuming to use tools such as Microsoft Visio or LaTeX to draw an FA diagram when compared to drawing it by hand and scanning it. Due to this, there is the need to develop a system that can take hand-drawn diagrams such as an FA and recognise it automatically. To conclude this thesis, a summary of the work presented in this thesis will be done and some limitations of the current system and some possible future work were discussed.

7.1 Conclusion

In this thesis, the issue of offline handwriting recognition has been discussed and addressed with a focus on FA diagrams for which segmentation into the components of the diagram was not known. Although there exist some techniques for recognition of structured diagrams such as FAs, flowcharts and UML diagrams, most of these techniques are based on online recognition approaches, grammar rules and editors. However, we wanted to have a flexible, time-saving and easy to use system. Our system should adapt to variation in writing styles from person to person, and deal with noise.

To achieve this, we introduced a multi-phase recognition approach. Each of the phases has been chosen to work best for its function.

This thesis presents a number of contributions to the fields of offline recognition and document image analysis. These contributions are stated below:

(i) Due to the lack of annotated data in the field of offline handwritten diagram recognition, we have created and annotated two offline datasets for FA diagrams. We showed that using these diagrams, the developed recognition system can achieve state-of-the-art results.

(ii) Due to the domain of this study, we adapted the thinning algorithm proposed in Boudaoud et al. [2015] for FA diagrams.

(iii) A new segmentation technique based on the bounding box approach was proposed. Using the proximity or closeness of a symbol that is wrongly segmented (i.e. a single symbol divided into two symbols during segmentation), we proposed a post-segmentation algorithm that joins these two symbols to form the original symbol.

(iv) The arrow detector and recogniser in Kara and Stahovich [2007] was extended in this study. The reason for extending the recogniser in Kara and Stahovich [2007] is to allow different shapes of arrow shafts and arrow drawn as two consecutive strokes to be recognised.

(v) We also generated the transition table of an input FA diagram, after the diagram has passed through all the recognition stages.

The aim of this thesis is to extract hand-drawn FA diagrams from paper documents. This aim is achieved by using a multi-phase recognition approach. Two databases containing FA image with unconstrained drawing styles were introduced in this study. These images can be used by offline recognition systems. The dataset is split into a training set and a test set. This is used to train and build an FA recognition system. A new segmentation method and arrow detection and recognition approach were introduced in this thesis.

Due to the nature of the datasets used in this study, the existing Zhang-Suen thinning algorithm had to be extended. In this regard, we adapted the algorithm proposed in Boudaoud et al. [2015] for FA diagrams in Section 4.5. BLSTM-RNN is used to separate the graphical components of an FA from the textual components. This is to ensure that the system is fast during processing. The separation approach was validated by testing the classifier on the IAM handwriting database and the FA images in

the created databases. The classifier returned a good result. This is presented in Section 5.2. Furthermore, segmentation is generally regarded as a cumbersome task during offline recognition. This thesis presented a bounding box segmentation approach in Section 5.4. This approach uses the characteristics of an FA diagram (i.e. the components that make up an FA diagram are connected and drawn close to each other). Due to difficulty generally encountered during image segmentation, an efficient post-segmentation algorithm was developed. The post-segmentation algorithm checks and fixes symbols that are wrongly segmented during segmentation. This is to increase the accuracy and result of the segmentation phase. Arrow symbols were detected using their structure. This is presented in Section 5.5. The classifier classifies an arrow as a symbol that contains a line (shaft) with two identified points (the beginning and end-point of the shaft) and a head with three identified points. The classifier is trained to interpret an arrow that is coming from nowhere to a circle is always the start arrow. Also, two concentric circles are always a final or accepting state as explained in Section 5.6. Tesseract OCR is used to recognise all the removed text and all unused strokes in Section 5.7. In addition, we introduced an approach that generates the transition table of the recognised diagram in Section 5.9. During the experiment, the system achieved a high recognition rate at both stroke and symbol recognition level.

The results obtained from our experiment in Chapter 6 were very good. We showed that diagrams and symbols can be successfully segmented and classified when sufficient data is available. In the cases of wrongly segmented symbols, the post-segmentation algorithm analyses what the symbol is and correctly segments it. Generally, a high confidence value was noticed despite the variation in handwriting. The recognition system was fast at about 0.8 seconds on average to recognise a diagram and its results were accurate.

7.2 Future Work

Research in the field of offline handwriting recognition is still behind when compared to online handwriting recognition. Given this, it will be interesting to further investigate some aspects dealt with in this thesis. The performance and accuracy of the recognition system during experiment and evaluation at both stroke and symbol level was good, even for badly drawn diagrams and texts.

Extending the recognition system to recognise other diagrams such as flowcharts, UML diagrams, and chemical structures will be worth investigating. This will allow wider use of such systems by users from different backgrounds with different needs.

Investigating an approach to automatically perform annotation of hand-drawn diagrams while recognising them at the same time (i.e. a parallel process) will be a good further research direction, as this will reduce the running time of the system. This can be developed within the recognition system i.e. a single system.

As discussed earlier, segmentation of hand-drawn diagrams using an offline approach is challenging. Multiple tracing of strokes for emphasis purposes is a common occurrence when writing or drawing. This is one of the reasons segmenting handwriting is still an open research problem. Developing segmentation algorithms for systems to recognise multiple tracing using an offline approach is still challenging. Proposing a segmentation algorithm that can perform on hand-drawn connected diagrams irrespective of the level of pen emphasis, and the type of diagram (i.e. FA, flowchart, UML diagram) will be worth investigating. Developing a text detector that can recognise simple mathematical writing such as subscripts and superscripts will improve the accuracy of the text detection phase. Also, such a detector can be used in other recognition systems.

Several research has been conducted on how to make the course content of formal languages and automata theory (FLAT) easier for learners and help learners to conceptualise the different topics in FLAT [Berque et al., 2001; Chesñevar et al., 2003; Zingaro, 2008]. The research conducted in Pillay [2010] attempted to find out the difficulties experienced by FLAT learners; they identified problem-solving as the major challenge. Sanders et al. [2015] studied the common errors made by students when designing FAs. FLAT tools used for visualization such as Java Formal Language and Automata Package (JFLAP) [Rodger and Finley, 2006] help learners get ideas of the given problem and also check their solutions. However, these tools do not help them to further develop their problem-solving skills to an acceptable stage. To further develop their skills, some learners work on extra problems with feedback at each stage. To assist learners, incorporating a grading system in the recognition system will be an interesting research focus. This can help give feedback to learners and instructors and serve as an academic teaching tool. The grading system will check if the generated transition table is correct before assigning a score to the diagram.

Due to the recent advances in deep neural networks (DNN), using DNN will definitely give better accuracy and output when compared to the multi-phase approach we have used [Yadav and Bethard, 2019]. A DNN self-training algorithm can be investigated for this purpose. Such algorithms can learn how to detect and recognise the components of an FA diagram on its own by looking and learning how users draw FA diagrams.

References

[Al-Helali and Mahmoud, 2017] Baligh M Al-Helali and Sabri A Mahmoud. Arabic online handwriting recognition (AOHR): a survey. *ACM Computing Surveys (CSUR)*, 50(3): 33, 2017.

[Albus et al., 2012] John Edward Albus, RH Anderson, JM Brayer, R DeMori, H-YF Feng, SL Horowitz, B Moayer, T Pavlidis, WW Stallings, PH Swain, et al. *Syntactic pattern recognition, applications*, volume 14. Springer Science & Business Media, 2012.

[Alghoul et al., 2018] Ahmed Alghoul, Sara Al Ajrami, Ghada Al Jarousha, Ghayda Harb, and Samy S. Abu-Naser. Email classification using artificial neural network. *International Journal of Academic Engineering Research (IJAER)*, 2(11):8–14, 2018.

[Almomani et al., 2013] Ammar Almomani, Brij Gupta, Samer Atawneh, Andrew Meulenberg, and Eman Almomani. A survey of phishing email filtering techniques. *IEEE Communications Surveys & Tutorials*, 15(4):2070–2090, 2013.

[Alvarado, 2007] Christine Alvarado. Multi-domain sketch understanding. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-1823-5. doi: 10.1145/1281500.1281518. URL http://doi.acm.org/10.1145/1281500.1281518.

[Alvarado and Davis, 2004] Christine Alvarado and Randall Davis. Sketchread: a multi-domain sketch recognition engine. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 23–32. ACM, 2004.

[Alvarado and Davis, 2006] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural computer-based sketching environment. In *ACM SIGGRAPH 2006 Courses*, pages 24–31. ACM, 2006.

[Alvaro et al., 2014] Francisco Alvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Offline features for classifying handwritten Math symbols with recurrent neural networks. In *22nd International Conference on Pattern Recognition*, pages 2944–2949. IEEE, 2014.

[Anquetil et al., 1999] Éric Anquetil, Bertrand Coüasnon, and Frédéric Dambreville. A symbol classifier able to reject wrong shapes for document recognition systems. In *International Workshop on Graphics Recognition*, pages 209–218. Springer, 1999.

[Auer et al., 2013] Christopher Auer, Christian Bachmaier, Franz Brandenburg, and Andreas Gleißner. Calibration in optical graph recognition. *Image-A: Applicable Mathematics in Image Engineering, 3 (5), 11-14*, 2013.

[Awal et al., 2011] Ahmad-Montaser Awal, Guihuan Feng, Harold Mouchere, and Christian Viard-Gaudin. First experiments on a new online handwritten flowchart database. In *Document Recognition and Retrieval XVIII*, pages 81–90. International Society for Optics and Photonics, 2011.

[Babalola, 2015] Olusola Tope Babalola. Automatic recognition and interpretation of finite state automata diagrams. Master's thesis, Stellenbosch: Stellenbosch University, 2015.

[Bengio et al., 2016] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[Berque et al., 2001] Dave Berque, David K Johnson, and Larry Jovanovic. Teaching theory of computation using pen-based computers and an electronic whiteboard. In *ACM SIGCSE Bulletin*, volume 33, pages 169–172. ACM, 2001.

[Birchfield, 2016] Stan Birchfield. *Image processing and analysis*. Cengage Learning, 2016.

[Bishop, 2006] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[Bishop et al., 2004] Christopher Bishop, Markus Svensen, and Goeffrey Hinton. Distinguishing text from graphics in on-line handwritten ink. In *Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 142–147. IEEE, 2004.

[Bisht, 2019] Ankit Bisht. ML | classification vs regression, 2019. URL `https://www.geeksforgeeks.org/ml-classification-vs-regression/`. Accessed: 10 October 2019.

[Blagojevic et al., 2011] Rachel Blagojevic, Beryl Plimmer, John Grundy, and Yong Wang. Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams. *Computers & Graphics*, 35(5):976–991, 2011.

[Bluche, 2015] Théodore Bluche. *Deep neural networks for large vocabulary handwritten text recognition*. PhD thesis, Universite Paris-Sud, 2015.

[Bott and Laviola Jr, 2015] Jared Bott and Joseph Laviola Jr. The WOZ recognizer: A wizard of Oz sketch recognition system. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(3):15, 2015.

[Boudaoud et al., 2015] Lynda Ben Boudaoud, Abderrahmane Sider, and Abdelkamel Tari. A new thinning algorithm for binary images. In *2015 3rd International Conference on Control, Engineering & Information Technology (CEIT)*, pages 1–6. IEEE, 2015.

[Boyat and Joshi, 2015] Ajay Kumar Boyat and Brijendra Kumar Joshi. A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489*, 2015.

[Bresler et al., 2014] Martin Bresler, Truyen Van Phan, Daniel Prusa, Masaki Nakagawa, and Václav Hlavác. Recognition system for on-line sketched diagrams. In *14th International Conference on Frontiers in Handwriting Recognition*, pages 563–568. IEEE, 2014.

[Bresler et al., 2015] Martin Bresler, Daniel Prusa, and Václav Hlavác. Detection of arrows in on-line sketched diagrams using relative stroke positioning. In *IEEE Winter Conference on Applications of Computer Vision*, pages 610–617. IEEE, 2015.

[Carabias, 2012] Daniel Martin Carabias. Analysis of image thresholding methods for their application to augmented reality environments. Master's thesis, Universidad Complutense De Madrid, 2012.

[Carbune et al., 2019] Victor Carbune, Pedro Gonnet, Thomas Deselaers, Henry A Rowley, Alexander Daryin, Marcos Calvo, Li-Lun Wang, Daniel Keysers, Sandro Feuz, and Philippe Gervais. Fast multi-language LSTM-based online handwriting recognition. *arXiv preprint arXiv:1902.10525*, 2019.

[Celebi and Aydin, 2016] Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms*. Springer, 2016.

[Cesarini et al., 1997] Francesca Cesarini, Enrico Francesconi, Marco Gori, Simone Marinai, JQ Sheng, and Giovanni Soda. A neural-based architecture for spot-noisy logo recognition. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition, 1997.*, volume 1, pages 175–179. IEEE, 1997.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3): 27, 2011.

[Chen et al., 2015a] Kai Chen, Zhi-Jie Yan, and Qiang Huo. A context-sensitive-chunk BPTT approach to training deep LSTM/BLSTM recurrent neural networks for offline handwriting recognition. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 411–415. IEEE, 2015a.

[Chen et al., 2014] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. *arXiv preprint arXiv:1412.7062*, 2014.

[Chen et al., 2015b] Quan Chen, Dapeng Shi, Guihuan Feng, Xiaoyan Zhao, and Bin Luo. On-line handwritten flowchart recognition based on logical structure and graph grammar. In *5th International Conference on Information Science and Technology (ICIST), 2015*, pages 424–429. IEEE, 2015b.

[Chesñevar et al., 2003] Carlos I Chesñevar, María L Cobo, and William Yurcik. Using theoretical computer simulators for formal languages and automata theory. *ACM SIGCSE Bulletin*, 35(2):33–37, 2003.

[Chung et al., 2005] Ronald Chung, Petrut Mirica, and Beryl Plimmer. InkKit: a generic design tool for the tablet PC. In *Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: making CHI natural*, pages 29–30. ACM, 2005.

[Cortes and Vapnik, 1995] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[Costagliola et al., 2005] Gennaro Costagliola, Vincenzo Deufemia, and Michele Risi. Sketch grammars: A formalism for describing and recognizing diagrammatic sketch languages. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition, 2005.*, pages 1226–1230. IEEE, 2005.

[Costagliola et al., 2007] Gennaro Costagliola, Vincenzo Deufemia, and Michele Risi. Using grammar-based recognizers for symbol completion in diagrammatic sketches. In *Ninth International Conference on Document Analysis and Recognition, 2007.*, volume 2, pages 1078–1082. IEEE, 2007.

[Dalal and Triggs, 2005] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[Damen and Hogg, 2009] Dima Damen and David C Hogg. Attribute multiset grammars for global explanations of activities. In *British Machine Vision Conference (BMVC)*, pages 1–11, 2009.

[De Marsico et al., 2019] Maria De Marsico, Gabriella Sanniti di Baja, and Ana Fred. *Pattern Recognition Applications and Methods: 7th International Conference, ICPRAM 2018, Funchal, Madeira, Portugal, January 16-18, 2018, Revised Selected Papers*, volume 11351. Springer, 2019.

[De Mulder et al., 2015] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

[DeepAI, 2019] DeepAI. Neural network, 2019. URL https://deepai.org/machine-learning-glossary-and-terms/neural-network. Accessed: 25 May 2020.

[Dekel et al., 2015] Tali Dekel, Shaul Oron, Michael Rubinstein, Shai Avidan, and William T Freeman. Best-buddies similarity for robust template matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2021–2029, 2015.

[Deng et al., 2000] Weian Deng, S Sitharama Iyengar, and Nathan E Brener. A fast parallel thinning algorithm for the binary image skeletonization. *The International Journal of High Performance Computing Applications*, 14(1):65–81, 2000.

[Deniziak and Michno, 2016] Stanislaw Deniziak and Tomasz Michno. Content based image retrieval using query by approximate shape. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 807–816. IEEE, 2016.

[Di Martino and Esposito, 2016] Beniamino Di Martino and Antonio Esposito. A rule-based procedure for automatic recognition of design patterns in UML diagrams. *Software: Practice and Experience*, 46(7):983–1007, 2016.

[Ding et al., 2009] Kai Ding, Guoqiang Deng, and Lianwen Jin. An investigation of imaginary stroke techinique for cursive online handwriting chinese character recognition. In *2009 10th International Conference on Document Analysis and Recognition*, pages 531–535. IEEE, 2009.

[Du et al., 2016] Jun Du, Zi-Rui Wang, Jian-Fang Zhai, and Jin-Shui Hu. Deep neural network based hidden Markov model for offline handwritten Chinese text recognition. In *23rd International Conference on Pattern Recognition (ICPR)*, pages 3428–3433. IEEE, 2016.

[Duan et al., 2010] Haibin Duan, Chunfang Xu, Senqi Liu, and Shan Shao. Template matching using chaotic imperialist competitive algorithm. *Pattern Recognition Letters*, 31 (13):1868–1875, 2010.

[Ebrahimzadeh and Jampour, 2014] Reza Ebrahimzadeh and Mahdi Jampour. Efficient handwritten digit recognition based on histogram of oriented gradients and SVM. *International Journal of Computer Applications*, 104(9), 2014.

[Elleuch et al., 2016] Mohamed Elleuch, Rania Maalej, and Monji Kherallah. A new design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition. *Procedia Computer Science*, 80:1712–1723, 2016.

[Essays, 2018] UK Essays. Pattern recognition and classification theory, 2018. URL https://www.ukessays.com/essays/engineering/

pattern-recognition-classification-2997.php?vref=1. Accessed: 24 May 2020.

[Evans et al., 1998] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Developing the UML as a formal modelling notation. In *Proc. UML98, LNCS*, volume 1618, 1998.

[Farahmand et al., 2013] Atena Farahmand, Hossein Sarrafzadeh, and Jamshid Shanbehzadeh. Document image noises and removal methods. In *Proceeding of International Multi-conference of Engineers and Computer Scientists*, pages 1–5, 2013.

[Firsov and Uustalu, 2014] Denis Firsov and Tarmo Uustalu. Certified CYK parsing of context-free languages. *Journal of Logical and Algebraic Methods in Programming*, 83 (5-6):459–468, 2014.

[Freeman and Plimmer, 2007] Isaac J Freeman and Beryl Plimmer. Connector semantics for sketched diagram recognition. In *Proceedings of the Eight Australasian Conference on User Interface-Volume 64*, pages 71–78. Australian Computer Society, Inc., 2007.

[Frinken et al., 2014] Volkmar Frinken, Andreas Fischer, Markus Baumgartner, and Horst Bunke. Keyword spotting for self-training of BLSTM NN based handwriting recognition systems. *Pattern Recognition*, 47(3):1073–1082, 2014.

[Fu et al., 2019] Bo Fu, Xiaoyang Zhao, Chuanming Song, Ximing Li, and Xianghai Wang. A salt and pepper noise image denoising method based on the generative classification. *Multimedia Tools and Applications*, 78(9):12043–12053, 2019.

[Fu and Han, 2012] Zhaoxia Fu and Yan Han. A circle detection algorithm based on mathematical morphology and chain code. In *International Conference on Computing, Measurement, Control and Sensor Network*, pages 253–256. IEEE, 2012.

[Gandhi, 2016] Rohith Gandhi. Introduction to machine learning algorithms: Linear regression, 2016. URL https://towardsdatascience.com/. Accessed: 24 July 2019.

[Gehrke et al., 2009] Martin Gehrke, Karl-Heinz Steinke, and Robert Dzido. Writer recognition by characters, words and sentences. In *43rd Annual 2009 International Carnahan Conference on Security Technology*, pages 281–288. IEEE, 2009.

[Ghaddar and Naoum-Sawaya, 2018] Bissan Ghaddar and Joe Naoum-Sawaya. High dimensional data classification and feature selection using support vector machines. *European Journal of Operational Research*, 265(3):993–1004, 2018.

[Gonzalez et al., 2004] Rafael C Gonzalez, Richard Eugene Woods, and Steven L Eddins. *Digital image processing using MATLAB*. Pearson Education India, 2004.

[Grailu et al., 2009] Hadi Grailu, Mojtaba Lotfizad, and Hadi Sadoghi-Yazdi. An improved pattern matching technique for lossy/lossless compression of binary printed Farsi and Arabic textual images. *International Journal of Intelligent Computing and Cybernetics*, 2(1):120–147, 2009.

[Graves and Schmidhuber, 2009] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 545–552, 2009.

[Graves et al., 2009] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[Guo and Hall, 1989] Zicheng Guo and Richard W Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, 1989.

[Hajič and Pecina, 2017] Jan Hajič and Pavel Pecina. The muscima++ dataset for handwritten optical music recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 39–46. IEEE, 2017.

[Hajič Jr and Pecina, 2017] Jan Hajič Jr and Pavel Pecina. Detecting noteheads in handwritten scores with convnets and bounding box regression. *arXiv preprint arXiv:1708.01806*, 2017.

[Hall et al., 2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[Halpin et al., 2003] Terence Aidan Halpin, Ken Evans, and Pat Hallock. *Database modeling with Microsoft visio for enterprise architects*. Morgan Kaufmann, 2003.

[Hammond and Davis, 2005] Tracy Hammond and Randall Davis. LADDER, a sketching language for user interface developers. *Computers & Graphics*, 29(4):518–532, 2005.

[Hammond and Davis, 2006] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for UML class diagrams. In *ACM SIGGRAPH 2006 Courses*, page 25. ACM, 2006.

[He et al., 2017] Jun He, Dongliang Li, Bin Yang, Siming Cao, Bo Sun, and Lejun Yu. Multi view facial action unit detection based on CNN and BLSTM-RNN. In *12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 848–853. IEEE, 2017.

[He et al., 2010] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *European Conference on Computer Vision*, pages 1–14. Springer, 2010.

[Heath et al., 1997] Michael D Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin W Bowyer. A robust visual method for assessing the relative performance of edge-detection algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12): 1338–1359, 1997.

[Ho and Baird, 1997] Tin Kam Ho and Henry S. Baird. Large-scale simulation studies in image pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(10):1067–1079, 1997.

[Ho-Quang et al., 2014] Truong Ho-Quang, Michel RV Chaudron, Ingimar Samúelsson, Jóel Hjaltason, Bilal Karasneh, and Hafeez Osman. Automatic classification of UML class diagrams from images. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 399–406. IEEE, 2014.

[Hoang and Tabbone, 2010] Thai V Hoang and Salvatore Tabbone. Text extraction from graphical document images using sparse representation. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 143–150. ACM, 2010.

[Hu and Zanibbi, 2016] Lei Hu and Richard Zanibbi. Line-of-sight stroke graphs and parzen shape context features for handwritten math formula representation and symbol segmentation. In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 180–186. IEEE, 2016.

[Huang et al., 2012] Mengxing Huang, Wenjiao Yu, and Donghai Zhu. An improved image segmentation algorithm based on the Otsu method. In *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 135–139. IEEE, 2012.

[Huo et al., 2015] Qiang Huo, Zhi-Jie Yan, Yu Zhang, and Jian Xu. I-Vector based clustering training data in speech recognition, July 16 2015. US Patent App. 13/640,804.

[Jain et al., 2001] Anil Jain, Arun Ross, and Salil Prabhakar. Fingerprint matching using minutiae and texture features. In *Proceedings of the International Conference on Image Processing, 2001.*, volume 3, pages 282–285. IEEE, 2001.

[Jain et al., 2000] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1): 4–37, 2000.

[Jiang et al., 2016] Bo Jiang, Jin Tang, Aihua Zheng, and Bin Luo. Image representation and matching with geometric-edge random structure graph. *Pattern Recognition Letters*, 2016.

[Jianzhuang et al., 1991] Liu Jianzhuang, Li Wenqing, and Tian Yupeng. Automatic thresholding of gray-level pictures using two-dimension Otsu method. In *International Conference on Circuits and Systems*, pages 325–327. IEEE, 1991.

[Kanisha et al., 2018] B Kanisha, S Lokesh, Priyan Malarvizhi Kumar, P Parthasarathy, and Gokulnath Chandra Babu. Speech recognition with improved support vector machine using dual classifiers and cross fitness validation. *Personal and Ubiquitous Computing*, 22(5-6):1083–1091, 2018.

[Kara and Stahovich, 2007] Levent Burak Kara and Thomas F Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *ACM SIGGRAPH 2007 Courses*, page 17. ACM, 2007.

[Karasneh and Chaudron, 2013] Bilal Karasneh and Michel RV Chaudron. Extracting UML models from images. In *5th International Conference on Computer Science and Information Technology*, pages 169–178. IEEE, 2013.

[Keysers et al., 2016] Daniel Keysers, Thomas Deselaers, Henry Rowley, Li-Lun Wang, and Victor Carbune. Multi-language online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1180–1194, 2016.

[Kim and Kim, 2000] Whoi-Yul Kim and Yong-Sung Kim. A region-based shape descriptor using zernike moments. *Signal Processing: Image Communication*, 16(1-2):95–102, 2000.

[Kovalevsky, 2019] Vladimir Kovalevsky. Image segmentation and connected components. In *Modern Algorithms for Image Processing*, pages 167–186. Springer, 2019.

[Kpalma and Ronsin, 2007] Kidiyo Kpalma and Joseph Ronsin. An Overview of Advances of Pattern Recognition Systems in Computer Vision. In Prof. Goro Obinata and Dr. Ashish Dutta, editors, *Vision Systems*, page 26. Advanced Robotic Systems, May 2007. URL `https://hal.archives-ouvertes.fr/hal-00143842`. 26 pages.

[Krattenthaler et al., 1994] Werner Krattenthaler, Konrad J Mayer, and Michael Zeiller. Point correlation: a reduced-cost template matching technique. In *Proceedings of the ICIP-94. Image Processing International Conference*, volume 1, pages 208–212. IEEE, 1994.

[Krizhevsky et al., 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[Kurnia et al., 2017] Rahmadi Kurnia, Melia Asmita, and Ikhwana Elfitri. Object detection on hindered condition by using chain code-based angle detection. In *Proceedings of the 2017 International Conference on Telecommunications and Communication Engineering*, pages 50–56. ACM, 2017.

[Lan-yi and Shu-guang, 2013] Xia Lan-yi and Dai Shu-guang. Circle and circular arc detection algorithm research based on freeman chain code. In *2013 IEEE 4th International Conference on Electronics Information and Emergency Communication*, pages 230–233. IEEE, 2013.

[Landay and Myers, 1995] James A Landay and Brad A Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the Special Interest Group on Computer-Human Interaction (SIGCHI) Conference on Human Factors in Computing Systems*, pages 43–50. ACM Press/Addison-Wesley Publishing Co., 1995.

[Landay and Myers, 2001] James A Landay and Brad A Myers. Sketching interfaces: Toward more human interface design. *Computer*, 34(3):56–64, 2001.

[Lank et al., 2000] Edward Lank, Jeb S Thorley, and Sean Jy-Shyang Chen. An interactive system for recognizing hand drawn UML diagrams. In *Proceedings of the 2000 Conference of the Centre for Advanced Studies on Collaborative Research*, page 7. IBM Press, 2000.

[Le and Nakagawa, 2016] Anh Duc Le and Masaki Nakagawa. A system for recognizing online handwritten mathematical expressions by using improved structural analysis. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(4):305–319, 2016.

[LeCun et al., 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lee et al., 2001] Yongbum Lee, Takeshi Hara, Hiroshi Fujita, Shigeki Itoh, and Takeo Ishigaki. Automated detection of pulmonary nodules in helical CT images based on an improved template-matching technique. *IEEE Transactions on Medical Imaging*, 20(7): 595–604, 2001.

[Leichtle et al., 2017] Tobias Leichtle, Christian Geiß, Michael Wurm, Tobia Lakes, and Hannes Taubenböck. Unsupervised change detection in VHR remote sensing imagery–an object-based clustering approach in a dynamic urban environment. *International Journal of Applied Earth Observation and Geoinformation*, 54:15–27, 2017.

[Li et al., 2012] Jie Li, Yue-Ying Peng, Chang-An Yuan, Mo Lin, and Ren-Min Wang. Edge thinning based on mathematical morphology thinning algorithm. *Journal of Computer Applications*, 2, 2012.

[Li et al., 2018] Xiao-Hui Li, Fei Yin, and Cheng-Lin Liu. Printed/handwritten texts and graphics separation in complex documents using conditional random fields. In *13th IAPR International Workshop on Document Analysis Systems*, pages 145–150. IEEE, 2018.

[Li et al., 2004] Zhenjiang Li, Honggui Wan, Yuhu Shi, and Pingkai Ouyang. Personal experience with four kinds of chemical structure drawing software: review on ChemDraw, ChemWindow, ISIS/Draw, and ChemSketch. *Journal of Chemical Information and Computer Sciences*, 44(5):1886–1890, 2004.

[Liang and Hu, 2015] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375, 2015.

[Liaw et al., 2002] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[Liew et al., 2005] Alan Wee-Chung Liew, Hong Yan, and Mengsu Yang. Pattern recognition techniques for the emerging field of bioinformatics: A review. *Pattern Recognition*, 38 (11):2055–2073, 2005.

[Lin et al., 2000] James Lin, Mark W Newman, Jason I Hong, and James A Landay. DENIM: finding a tighter fit between tools and practice for Web site design. In *Proceedings of the Special Interest Group on Computer-Human Interaction (SIGCHI) Conference on Human Factors in Computing Systems*, pages 510–517. ACM, 2000.

[Lipton et al., 2015] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[Liu et al., 2012] Jiongxin Liu, Angjoo Kanazawa, David Jacobs, and Peter Belhumeur. Dog breed classification using part localization. In *European Conference on Computer Vision*, pages 172–185. Springer, 2012.

[Liwicki and Bunke, 2005] Marcus Liwicki and Horst Bunke. IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 956–961. IEEE, 2005.

[Ma and Guo, 2014] Yunqian Ma and Guodong Guo. *Support vector machines applications*, volume 649. Springer, 2014.

[Maggiori et al., 2016] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):645–657, 2016.

[Mahdavi et al., 2019] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchère, Christian Viard-Gaudin, and Utpal Garain. ICDAR 2019 CROHME+ TFD: Competition on recognition of handwritten mathematical expressions and typeset formula detection. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1533–1538. IEEE, 2019.

[Maheswari and Radha, 2010] D Maheswari and V Radha. Noise removal in compound image using median filter. *International Journal on Computer Science and Engineering*, 2 (04):1359–1362, 2010.

[Mahmoud and Assayony, 2018] Sabri A Mahmoud and Mohammed O Assayony. Arabic handwriting recognition system and method, December 25 2018. US Patent Application 10/163,019.

[Mahoney and Fromherz, 2002] James V Mahoney and Markus PJ Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *Association for the Advancement of Artificial Intelligence, Spring Symposium on Sketch Understanding*, pages 105–112, 2002.

[Mallick, 2015] Satya Mallick. OpenCV threshold (Python , C++), 2015. URL `https://www.learnopencv.com/opencv-threshold-python-cpp/`. Accessed: 16 September 2019.

[Mantas, 1986] J Mantas. An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430, 1986.

[Marriott, 1994] Kim Marriott. Constraint multiset grammars. In *Proceedings of the IEEE Symposium on Visual Languages, 1994.*, pages 118–125. IEEE, 1994.

[Marti and Bunke, 2001] U-V Marti and Horst Bunke. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. In *Hidden Markov models: applications in computer vision*, pages 65–90. World Scientific, 2001.

[Marti and Bunke, 2002] U-V Marti and Horst Bunke. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.

[Mas et al., 2007] Joan Mas, Joaquim A Jorge, Gemma Sanchez, and Josep Llados. Representing and parsing sketched symbols using adjacency grammars and a grid-directed

parser. In *International Workshop on Graphics Recognition*, pages 169–180. Springer, 2007.

[Memon et al., 2020] Jamshed Memon, Maira Sami, and Rizwan Ahmed Khan. Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr). *arXiv preprint arXiv:2001.00139*, 2020.

[Messina and Louradour, 2015] Ronaldo Messina and Jerome Louradour. Segmentation-free handwritten Chinese text recognition with LSTM-RNN. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 171–175. IEEE, 2015.

[Mirkin, 2016] Boris Mirkin. *Clustering: a data recovery approach*. Chapman and Hall/CRC, 2016.

[Miyao and Maruyama, 2012] Hidetoshi Miyao and Rei Maruyama. On-line handwritten flowchart recognition, beautification and editing system. In *International Conference on Frontiers in Handwriting Recognition (ICFHR), 2012*, pages 83–88. IEEE, 2012.

[Moghaddam and Cheriet, 2012] Reza Farrahi Moghaddam and Mohamed Cheriet. AdOtsu: An adaptive and parameterless generalization of Otsu's method for document image binarization. *Pattern Recognition*, 45(6):2419–2431, 2012.

[Nasien et al., 2010] Dewi Nasien, Habibollah Haron, and Siti Sophiayati Yuhaniz. Support vector machine (SVM) for English handwritten character recognition. In *2010 Second International Conference on Computer Engineering and Applications*, volume 1, pages 249–252. IEEE, 2010.

[Nelligan et al., 2015] Trevor Nelligan, Seth Polsley, Jaideep Ray, Michael Helms, Julie Linsey, and Tracy Hammond. Mechanix: a sketch-based educational interface. In *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, pages 53–56. ACM, 2015.

[Otsu, 1979] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[Ouyang and Davis, 2009] Tom Y Ouyang and Randall Davis. A visual approach to sketched symbol recognition. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1463–1468, 2009.

[Ouyang and Davis, 2011] Tom Y Ouyang and Randall Davis. Chemink: a natural real-time recognition system for chemical drawings. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, pages 267–276. ACM, 2011.

[Ozawa et al., 2009] Seiichi Ozawa, Asim Roy, and Dmitri Roussinov. A multitask learning model for online pattern recognition. *IEEE Transactions on Neural Networks*, 20(3): 430–445, 2009.

[Pan et al., 2016] Jianjia Pan, Xian-Wei Zheng, Li Sun, Li-Na Yang, Yu-Long Wang, Hui-Wu Luo, and Patrick Shen-Pei Wang. Image segmentation based on 2D Otsu and simplified swarm optimization. In *International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 2, pages 1026–1030. IEEE, 2016.

[Parkhi et al., 2012] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505. IEEE, 2012.

[Parkhi et al., 2015] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

[Patterson and Gibson, 2017] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. O'Reilly Media, Inc., 2017.

[Perez and Wang, 2017] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[Pillay, 2010] Nelishia Pillay. Learning difficulties experienced by students in a course on formal languages and automata theory. *ACM SIGCSE Bulletin*, 41(4):48–52, 2010.

[Plamondon and Srihari, 2000] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[Poznanski and Wolf, 2016] Arik Poznanski and Lior Wolf. CNN-N-Gram for handwriting word recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2305–2314, 2016.

[Prasad et al., 2018] Puja S Prasad, B Sunitha Devi, and Rony Preetam. Image enhancement for fingerprint recognition using Otsu's method. In *International Conference on Communications and Cyber Physical Engineering 2018*, pages 269–277. Springer, 2018.

[Ptak et al., 2017] Roman Ptak, Bartosz Żygadło, and Olgierd Unold. Projection–based text line segmentation with a variable threshold. *International Journal of Applied Mathematics and Computer Science*, 27(1):195–206, 2017.

[Rashid et al., 2011] Sheikh Faisal Rashid, Faisal Shafait, and Thomas M Breuel. An evaluation of HMM-based techniques for the recognition of screen rendered text. In *2011 International Conference on Document Analysis and Recognition*, pages 1260–1264. IEEE, 2011.

[Rastegari et al., 2016] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[Refaat et al., 2008] Khaled S Refaat, Wael N Helmy, AbdelRahman H Ali, Mohamed S AbdelGhany, and Amir F Atiya. A new approach for context-independent handwritten offline diagram recognition using support vector machines. In *IEEE International Joint Conference on Neural Networks, IJCNN 2008*, pages 177–182. IEEE, 2008.

[Rehman and Saba, 2014] Amjad Rehman and Tanzila Saba. Neural networks for document image preprocessing: state of the art. *Artificial Intelligence Review*, 42(2):253–273, 2014.

[Reiher et al., 1996] Eric Reiher, Ying Li, Vince Delle Donne, Marc Lalonde, Cameron Hayne, and Chona Zhu. A system for efficient and robust map symbol recognition. In *Proceedings of the 13th International Conference on Pattern Recognition, 1996.*, volume 3, pages 783–787. IEEE, 1996.

[Ren and Ganapathy, 2019] Guohua Ren and Viswanath Ganapathy. Recognition of online handwriting with variability on smart devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7605–7609. IEEE, 2019.

[Richardson et al., 2015] Fred Richardson, Douglas Reynolds, and Najim Dehak. Deep neural network approaches to speaker and language recognition. *IEEE Signal Processing Letters*, 22(10):1671–1675, 2015.

[Robert et al., 2003] Fisher Robert, Perkins Simon, Walker Ashley, and Wolfart Erik. Thinning, 2003. URL `https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm`. Accessed: 24 May 2019.

[Rodger and Finley, 2006] Susan H Rodger and Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[Roudaki et al., 2016] Amin Roudaki, Jun Kong, and Kang Zhang. Specification and discovery of web patterns: a graph grammar approach. *Information Sciences*, 328:528–545, 2016.

[Roy et al., 2016] Partha Pratim Roy, Ayan Kumar Bhunia, Ayan Das, Prasenjit Dey, and Umapada Pal. HMM-based indic handwritten word recognition using zone segmentation. *Pattern Recognition*, 60:1057–1075, 2016.

[Roy et al., 2018] Swalpa Kumar Roy, Bhabatosh Chanda, Bidyut B Chaudhuri, Dipak Kumar Ghosh, and Shiv Ram Dubey. Local morphological pattern: A scale space shape descriptor for texture classification. *Digital Signal Processing*, 82:152–165, 2018.

[Samanta et al., 2018] Oendrila Samanta, Anandarup Roy, Swapan K Parui, and Ujjwal Bhattacharya. An HMM framework based on spherical-linear features for online cursive handwriting recognition. *Information Sciences*, 441:133–151, 2018.

[Sanders et al., 2015] Ian Sanders, Colin Pilkington, and Wynand van Staden. Errors made by students when designing finite automata. In *44th Annual SACLA Conference*, volume 44, pages 110–118, 2015.

[Sankaranarayanan et al., 2016] Swami Sankaranarayanan, Azadeh Alavi, Carlos D Castillo, and Rama Chellappa. Triplet probabilistic embedding for face verification and clustering. In *IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–8. IEEE, 2016.

[Schroff et al., 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[Schubert et al., 2015] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. A framework for clustering uncertain data. *Proceedings of the VLDB Endowment*, 8(12):1976–1979, 2015.

[Schuster and Paliwal, 1997] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[Sha et al., 2016] Chunshi Sha, Jian Hou, and Hongxia Cui. A robust 2D Otsus thresholding method in image segmentation. *Journal of Visual Communication and Image Representation*, 41:339–351, 2016.

[Simistira et al., 2015] Fotini Simistira, Vassilis Katsouros, and George Carayannis. Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars. *Pattern Recognition Letters*, 53:85–92, 2015.

[Singh et al., 2016] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315. IEEE, 2016.

[Smith, 2007] Ray Smith. An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[Smithies et al., 1999] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Graphics Interface*, volume 99, pages 84–91, 1999.

[Sonka et al., 2014] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[Spanhol et al., 2015] Fabio A Spanhol, Luiz S Oliveira, Caroline Petitjean, and Laurent Heutte. A dataset for breast cancer histopathological image classification. *IEEE Transactions on Biomedical Engineering*, 63(7):1455–1462, 2015.

[Stoller, 1973] M Stoller. Optical character recognition system, March 27 1973. US Patent 3,723,970.

[Strobbe et al., 2015] Tiemen Strobbe, Pieter Pauwels, Ruben Verstraeten, Ronald De Meyer, and Jan Van Campenhout. Toward a visual approach in the exploration of shape grammars. *AI EDAM*, 29(4):503–521, 2015.

[Stuckelberg and Doermann, 1999] Marc Vuilleumier Stuckelberg and David Doermann. On musical score recognition using probabilistic reasoning. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition, 1999*, pages 115–118. IEEE, 1999.

[Sun and Neuvo, 1994] Tong Sun and Yrjö Neuvo. Detail-preserving median based filters in image processing. *Pattern Recognition Letters*, 15(4):341–347, 1994.

[Tang et al., 2014] Peng Tang, Siu Cheung Hui, and Chi-Wing Fu. Connected bond recognition for handwritten chemical skeletal structural formulas. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 122–127. IEEE, 2014.

[Tappert et al., 1990] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[Team, 2015] GraphML Team. The GraphML file format. *Homepage: http://graphml. graphdrawing. org*, 2015.

[Tolosana et al., 2018] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, and Javier Ortega-Garcia. Exploring recurrent neural networks for online handwritten signature biometrics. *IEEE Access*, 6:5128–5138, 2018.

[Tong and Koller, 2001] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2(Nov):45–66, 2001.

[Venguerov and Cunningham, 1998] Mark Venguerov and Pádraig Cunningham. Generalised syntactic pattern recognition as a unifying approach in image analysis. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 913–920. Springer, 1998.

[VenkateswaraRao et al., 2011] N VenkateswaraRao, A Srikrishna, B RaveendraBabu, and Rama Mohan Babu. An efficient feature extraction and classification of handwritten digits using neural networks. *International Journal of Computer Science, Engineering and Applications*, 1(5):47, 2011.

[Wang et al., 2016] Peng Wang, Bo Xu, Jig Xu, Guanhua Tian, Cheng-Lin Liu, and Hongwei Hao. Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification. *Neurocomputing*, 174:806–814, 2016.

[Wang et al., 2019] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1328–1338, 2019.

[Wang et al., 2014] Xiaolong Wang, Vincent Ly, Scott Sorensen, and Chandra Kambhamettu. Dog breed classification via landmarks. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 5237–5241. IEEE, 2014.

[Witten et al., 2016] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[Wong et al., 2016] Chin Yeow Wong, Guannan Jiang, Md Arifur Rahman, Shilong Liu, Stephen Ching-Feng Lin, Ngaiming Kwok, Haiyan Shi, Ying-Hao Yu, and Tonghai Wu. Histogram equalization and optimal profile compression based approach for colour image enhancement. *Journal of Visual Communication and Image Representation*, 38:802–813, 2016.

[Wu et al., 2015] Jie Wu, Changhu Wang, Liqing Zhang, and Yong Rui. Offline sketch parsing via shapeness estimation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[Xu et al., 2014] Liang Xu, Fei Yin, Qiu-Feng Wang, and Cheng-Lin Liu. An over-segmentation method for single-touching chinese handwriting with learning-based filtering. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(1): 91–104, 2014.

[Xu et al., 2011] Xiangyang Xu, Shengzhou Xu, Lianghai Jin, and Enmin Song. Characteristic analysis of Otsu threshold and its applications. *Pattern Recognition Letters*, 32(7): 956–961, 2011.

[Yadav and Bethard, 2019] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. *arXiv preprint arXiv:1910.11470*, 2019.

[Yang and Yan, 2000] Yibing Yang and Hong Yan. An adaptive logical method for binarization of degraded document images. *Pattern Recognition*, 33(5):787–807, 2000.

[Yao and Pakzad, 2012] Ruigen Yao and Shamim N Pakzad. Auto-regressive statistical pattern recognition algorithms for damage detection in civil structures. *Mechanical Systems and Signal Processing*, 31:355–368, 2012.

[Yin et al., 2013] Fei Yin, Qiu-Feng Wang, Xu-Yao Zhang, and Cheng-Lin Liu. ICDAR 2013 Chinese handwriting recognition competition. In *12th International Conference on Document Analysis and Recognition*, pages 1464–1470. IEEE, 2013.

[Yin et al., 2015] Xu-Cheng Yin, Wei-Yi Pei, Jun Zhang, and Hong-Wei Hao. Multi-orientation scene text detection with adaptive clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1930–1937, 2015.

[Yuan et al., 2017] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.

[Yuan et al., 2008] Zhenming Yuan, Hong Pan, and Liang Zhang. A novel pen-based flowchart recognition system for programming teaching. In *Advances in Blended Learning*, pages 55–64. Springer, 2008.

[Zamora-Martinez et al., 2014] Francisco Zamora-Martinez, Volkmar Frinken, Salvador España-Boquera, Maria Jose Castro-Bleda, Andreas Fischer, and Horst Bunke. Neural network language models for off-line handwriting recognition. *Pattern Recognition*, 47(4):1642–1652, 2014.

[Zhang and Lu, 2002] Dengsheng Zhang and Guojun Lu. A comparative study of three region shape descriptors. *DICTA2002: Digital Image Computing Techniques and Applications*, 2122, 2002.

[Zhang et al., 2018] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018.

[Zhang and Suen, 1984] TY Zhang and Ching Y Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

[Zhang et al., 2017] Xu-Yao Zhang, Yoshua Bengio, and Cheng-Lin Liu. Online and offline handwritten Chinese character recognition: A comprehensive study and new benchmark. *Pattern Recognition*, 61:348–360, 2017.

[Zhong et al., 2015] Zhuoyao Zhong, Lianwen Jin, and Zecheng Xie. High performance offline handwritten Chinese character recognition using googlenet and directional feature maps. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 846–850. IEEE, 2015.

[Zhou et al., 2015] Pan Zhou, Hui Jiang, Li-Rong Dai, Yu Hu, and Qing-Feng Liu. State-clustering based multiple deep neural networks modeling approach for speech recognition. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(4):631–642, 2015.

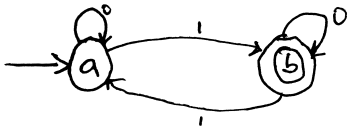
[Zingaro, 2008] Daniel Zingaro. Another approach for resisting student resistance to formal methods. *ACM SIGCSE Bulletin*, 40(4):56–57, 2008.

Appendix A

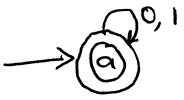
Sample Data

A.1 Synthetic Data

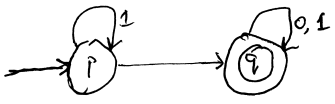
In this section, we illustrate some of the input diagrams.



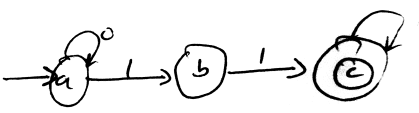
(a) FA drawn by student A



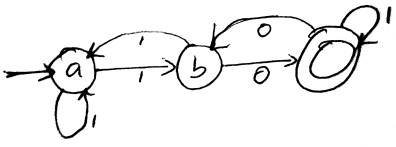
(b) FA drawn by student B



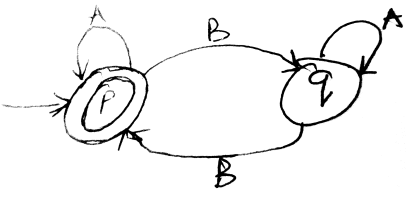
(c) FA drawn by student C



(d) FA drawn by student D

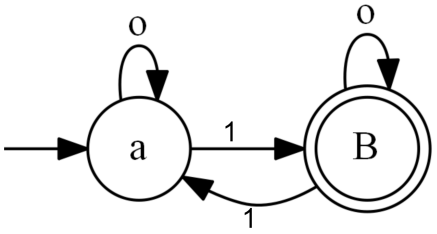


(e) FA drawn by student E

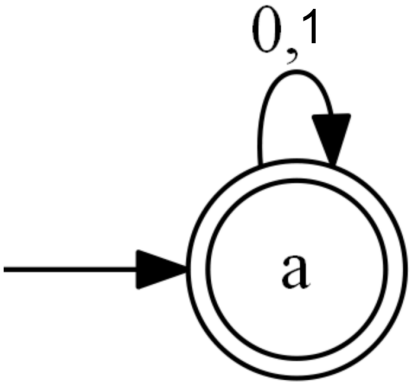


(f) FA drawn by student F

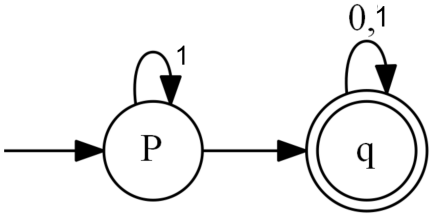
Figure A.1: Sample binarised input images



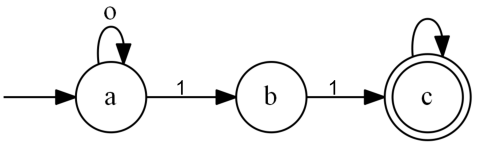
(a) FA diagram 1



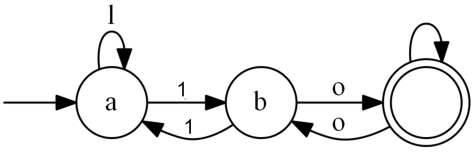
(b) FA diagram 2



(c) FA diagram 3



(d) FA diagram 4



(e) FA diagram 5

Figure A.2: Recognised FA diagrams

A.2 Raw Data

Figures A.3 and A.4 shows the FA we took from examination papers.

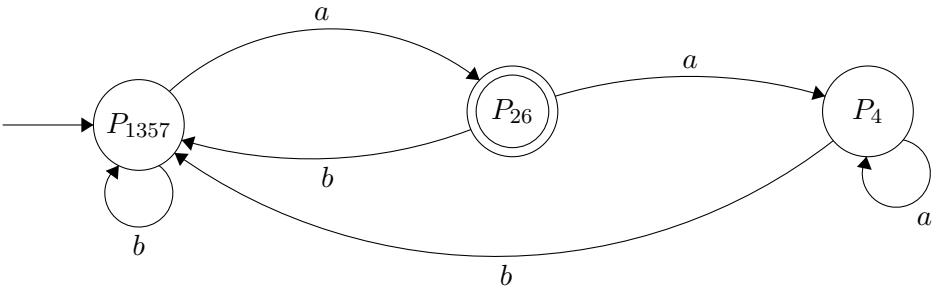


Figure A.3: LaTeX generated version of the first FA

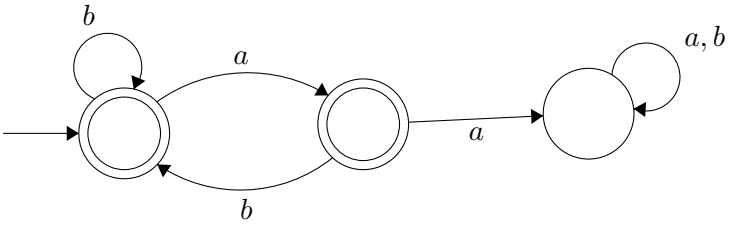
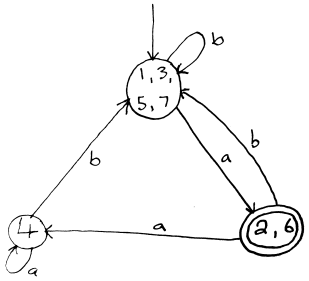
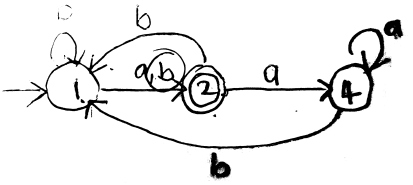


Figure A.4: LaTeX generated version of the second FA

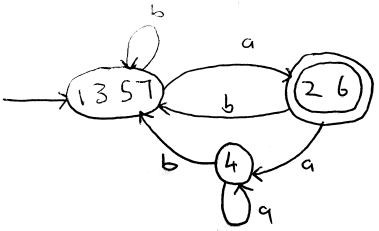
Below we present the binarised versions of Figures A.3 and A.4 as they were originally captured from the examination papers. We only present a few of the diagrams. There was no criterion for selecting the presented diagrams.



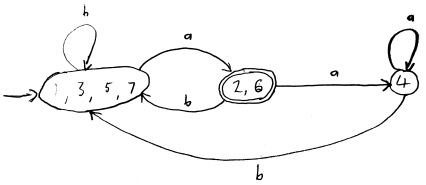
(a) FA diagram taken from Paper A



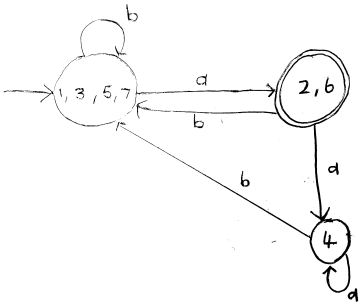
(b) FA diagram taken from Paper B



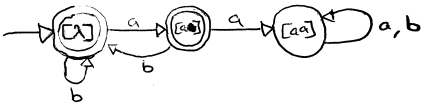
(c) FA diagram taken from Paper C



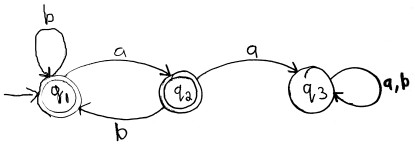
(d) FA diagram taken from Paper D



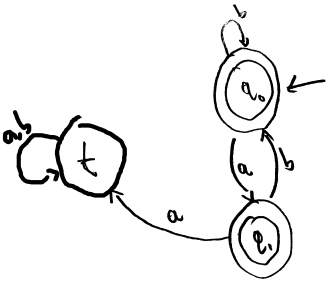
(e) FA diagram taken from Paper E



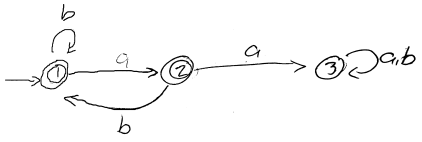
(f) FA diagram taken from Paper F



(g) FA diagram taken from Paper G



(h) FA diagram taken from Paper H



(i) FA diagram taken from Paper I

Figure A.5: Sample binarised input images

Appendix B

Participants Consent Letter

B.1 Student Letter



University of the Witwatersrand, Johannesburg

School of Computer Science and Applied Mathematics

Consent Form

Protocol Number: H17/04/01

Project title: Extracting Finite Automata from Hand-drawn Images

(i) I have read and had explained to me by Kehinde Aruleba the information sheet relating to this study.

(ii) I have had explained to me the purpose of the study and what will be required of me, and any questions have been answered to my satisfaction. I agree to the arrangements described in the information sheet in so far as they relate to my participation.

(iii) I understand that my participation is entirely voluntary and that I have the right to not participate in this study.

(iv) I have received a copy of this Consent Form and of the accompanying Information Sheet.

Name:
Signed:
Date:

B.2 Ethics Certificate

Research Office

HUMAN RESEARCH ETHICS COMMITTEE (NON-MEDICAL)
R14/49 Aruleba

CLEARANCE CERTIFICATE	**PROTOCOL NUMBER: H17/04/01**
PROJECT TITLE	Extracting finite automata from hand drawn images
INVESTIGATOR(S)	Mr K Aruleba
SCHOOL/DEPARTMENT	Computer Science & Applied Mathematics/
DATE CONSIDERED	21 April 2017
DECISION OF THE COMMITTEE	Approved
EXPIRY DATE	19 June 2020

DATE 20 June 2017 **CHAIRPERSON** pp (Professor J Knight)

cc: Supervisor : Professors S Ewert, I Sanders and Dr M Raborife

DECLARATION OF INVESTIGATOR(S)

To be completed in duplicate and **ONE COPY** returned to the Secretary at Room 10004, 10th Floor, Senate House, University. Unreported changes to the application may invalidate the clearance given by the HREC (Non-Medical)

I/We fully understand the conditions under which I am/we are authorized to carry out the abovementioned research and I/we guarantee to ensure compliance with these conditions. Should any departure to be contemplated from the research procedure as approved I/we undertake to resubmit the protocol to the Committee. **I agree to completion of a yearly progress report.**

Signature **Date** 21 / 06 / 2017

PLEASE QUOTE THE PROTOCOL NUMBER ON ALL ENQUIRIES

Appendix C

Data Annotation Tool

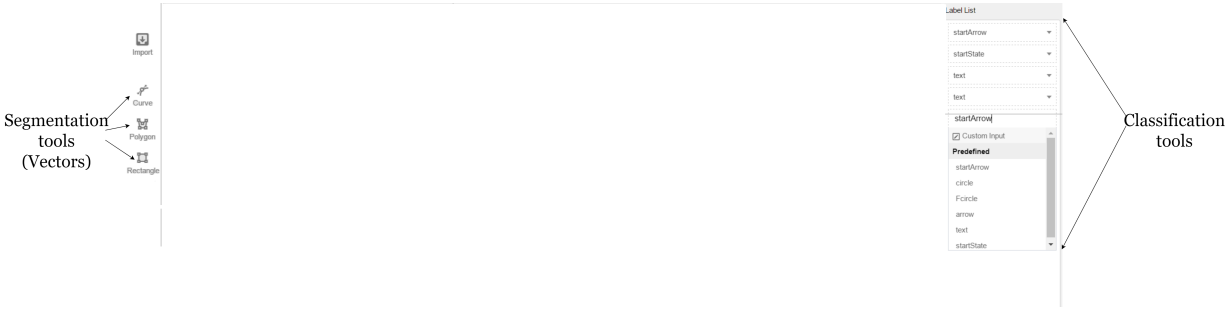


Figure C.1: Screenshot of the application for data annotation, including the annotation tools.

The next interface shows the annotation tool at run-time.

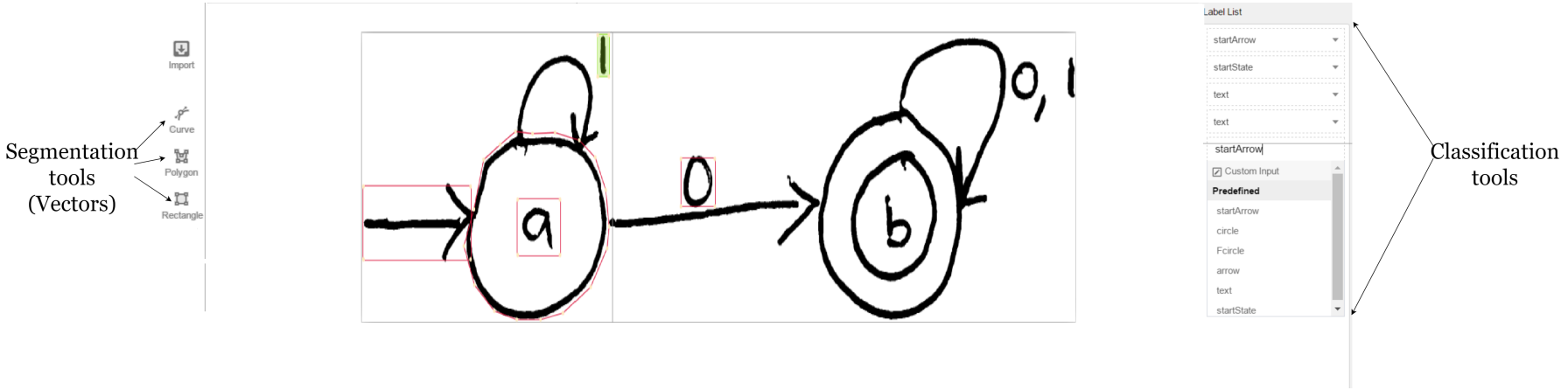


Figure C.2: Screenshot of the application for data annotation showing the annotation of some individual entities.