

Arabic Handwriting Recognition

Peter Burrow



Master of Science
School of Informatics
University of Edinburgh

2004

Abstract

This thesis explores a number of different techniques for use in the field of Arabic Handwriting Recognition. A review of previous work in the field is conducted, and then various techniques are explored in the context of classifying town names from the IFN/ENIT database. A baseline-finding algorithm using Principal Components Analysis is implemented, and the change in performance from reducing the influence of certain word features is also demonstrated. Several simple methods of town name classification are investigated, including a scheme using *Tangent Features*. These model the variations in the training examples in order to improve generalisation, and perform with 94% accuracy on a small 10-class lexicon. Moment invariants are considered as useful features for classification, but fail to surpass the performance of simpler methods. An approach where town names are split into parts and traced to recover temporal information is conceived, and found to have encouraging performance and several useful properties.

Acknowledgements

I would like to thank my supervisor, Louis Atallah, for his help and advice throughout this project. Our discussions proved to be both productive and enjoyable.

I would also like to thank Judy Burrow for her typing assistance, and the Spamlist for their omnipresent support and proof-reading skills.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Peter Burrow)

Table of Contents

1	Introduction	1
1.1	Background	1
1.1.1	Handwriting Recognition	1
1.1.2	Arabic	2
1.2	Previous work	3
1.2.1	An Overview of Off-line Handwriting Recognition	3
1.2.2	Arabic Cursive text	6
1.2.3	Preprocessing	8
1.3	Database	9
1.4	Outline of Forthcoming Chapters	10
2	Baseline	11
2.1	Introduction	11
2.2	Previous Work	12
2.3	Principal Components Analysis	14
2.3.1	Implementation	14
2.3.2	Results	15

2.4	Improved PCA	16
2.4.1	Implementation	17
2.4.2	Results	18
2.5	Discussion	19
3	Nearest Neighbour & Tangent Features	21
3.1	Introduction	21
3.2	Nearest Neighbour	22
3.2.1	Implementation	22
3.2.2	Results	23
3.3	Adding Tangent Features	24
3.3.1	Implementation	25
3.3.2	Results	26
3.4	Discussion	26
3.4.1	Nearest Neighbour	26
3.4.2	Tangent Features	27
4	Centroid & Moments	28
4.1	Introduction	28
4.2	Previous Work	29
4.3	Radial Method	30
4.3.1	Implementation	31
4.3.2	Results	32
4.4	Zernike Moments	32

4.4.1	Implementation	33
4.4.2	Results	34
4.5	Discussion	34
5	Tracing Approach	36
5.1	Introduction	36
5.2	Previous Work	37
5.3	Initial Method	38
5.3.1	Tracing	38
5.3.2	Feature Extraction	40
5.3.3	Classification	42
5.3.4	Results	42
5.4	Improved Method	43
5.4.1	Tracing and Feature Extraction	43
5.4.2	Classification	44
5.4.3	Results	47
5.5	Discussion	48
6	Discussion	50
6.1	Introduction	50
6.2	Summary of Results	51
6.3	Baseline Estimation	52
6.4	Tangent Features	52
6.5	Moment Invariants	53

6.6	Tracing Approach	53
6.7	Contribution to the field	54
6.8	Conclusions	55
6.9	Future Directions	56
	Bibliography	57

List of Figures

1.1	A Tunisian town name written in Arabic.	3
1.2	Different forms of the Arabic letter “A’in”.	4
2.1	Some features of Arabic words.	12
2.2	Projection histogram to find baseline position.	13
2.3	Some example results of the basic PCA method.	16
2.4	The influence of word features on the baseline.	17
2.5	Examples of improved PCA performance.	19
3.1	A Tunisian town name resized to 32×128 pixels.	23
3.2	Mean and Eigenvalues of town name transformations.	26
4.1	Example of the Radial Method.	31
5.1	Splitting a town name into POWs.	37
5.2	The basic tracing process.	39
5.3	Tracing at junctions.	40
5.4	Feature vector formation.	41
5.5	Confusion matrix for the initial tracing method.	42

5.6	Confusion matrix for the improved tracing method.	47
5.7	The tracing algorithm resolving part-of-word overlap.	48

List of Tables

2.1	Performance of PCA baseline estimation vs. other methods. . . .	15
2.2	Performance of PCA improvements.	18
3.1	Performance of simple Nearest Neighbour models.	23
4.1	Performance of the Radial Method on the small testset.	32
4.2	Performance of different orders of Zernike moments.	34
6.1	Summary of results on the small testset using different methods. .	51
6.2	Summary of results on the full database using different methods. .	51
6.3	Summary of best baseline estimation performance.	52

Chapter 1

Introduction

1.1 Background

1.1.1 Handwriting Recognition

Automated recognition of text has been an active subject of research since the early days of computers. A 1972 survey cites nearly 130 works on the subject [20]. Despite the age of the subject, it remains one of the most challenging and exciting areas of research in computer science. In recent years it has grown into a mature discipline, producing a huge body of work.

Despite long standing predictions that handwriting, and even paper itself, would become obsolete in the age of the digital computer, both persist. Whilst the computer has hugely simplified the process of producing printed documents, the convenience of a pen and paper still makes it the natural medium for many important tasks.

A brief survey of students in any lecture theatre will confirm the dominance of handwritten notes over those typing on laptops. However, the ease and convenience of having information in digital form provides a powerful incentive to find a way of quickly converting handwritten text into its digital equivalent.

Handwriting recognition can be defined as the task of transforming text represented in the spatial form of graphical marks into its symbolic representation.

Not only is this useful for making digital copies of handwritten documents, but also in many automated processing tasks, such as automatic mail sorting or cheque processing. In automated mail sorting, letters are directed to the correct location by recognition of the handwritten address. Similarly, cheque processing involves recognising the words making up the cheque amount.

The field of handwriting recognition can be split into two different approaches. The first of these, *on-line*, deals with the recognition of handwriting captured by a tablet or similar touch-sensitive device, and uses the digitised trace of the pen to recognise the symbol. In this instance the recogniser will have access to the x and y coordinates as a function of time, and thus has temporal information about how the symbol was formed.

The second approach concentrates on the recognition of handwriting in the form of an image, and is termed *off-line*. In this instance only the completed character or word is available. It is this off-line approach that will be taken in this report.

1.1.2 Arabic

Arabic is written by more than 100 million people, in over 20 different countries. The Arabic script evolved from a type of Aramaic, with the earliest known document dating from 512 AD. The Aramaic language has fewer consonants than Arabic, so new letters were created around the 7th century by adding dots to existing letters. There are therefore several letters differing only by a single dot. Other small marks (diacritics) are used to indicate short vowels, but are often not used.

A summary of the features of Arabic writing appears below.

1. Arabic text, both handwritten and printed, is *cursive*. The letters are joined together along a writing line (see Figure 1.1). This is similar to Latin ‘joined

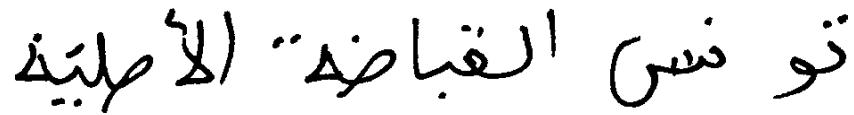

 The image shows a handwritten Arabic phrase in a cursive style. The text is written from right to left. The characters are connected, but the overall style is somewhat clear and legible. The phrase is "تونس القباضة الأملية".

Figure 1.1: A Tunisian town name written in Arabic.

up' handwriting, which is also cursive, but in which the characters are easier to separate.

2. In contrast to Latin text, Arabic is written right to left, rather than left to right. This is perhaps more significant for a human reader rather than a computer, since the computer can simply flip the images.
3. More importantly from the point of view of automated recognition, Arabic contains dots and other small marks that can change the meaning of a word, and need to be taken into account by any computerised recognition system. Often the diacritic marks representing vowels are left out, and the word must be identified from its context.
4. The shapes of the letters differ depending on whereabouts in the word they are found. The same letter at the beginning and end of a word can have a completely different appearance as shown in Figure 1.2. Along with the dots and other marks representing vowels, this makes the effective size of the alphabet about 160 characters.

1.2 Previous work

This review concentrates on off-line methods of recognition, since that is the form of recognition used in this project.

1.2.1 An Overview of Off-line Handwriting Recognition

Most of the work on handwriting recognition to date has been on Latin text. Possibly the most important distinction to make among off-line methods of recogni-

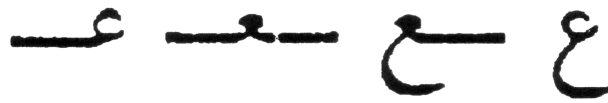


Figure 1.2: *Different forms of the Arabic letter "A'in". From left to right - beginning, middle, end, isolated. From Amin [1].*

tion is on the subject of segmentation. Methods can be divided into those which first segment the word to be recognised, and those that use the whole word. Since most of the previous work has focused on Latin text, which is easier to segment, the former approach has been the most popular. The following is a brief taste of work in the field of handwriting recognition. More detailed and comprehensive surveys of the subject can be found in [39, 35, 41].

1.2.1.1 Segmentation-based Methods

These methods take the approach of splitting a word into segments to be recognised, using a segmentation algorithm. There have been various attempts to reliably segment characters (e.g. Casey and Lecolinet [4]), but many have imperfections. Humans can easily segment characters by first recognising them. Whilst some methods try to combine segmentation with recognition (e.g. El-Dabi et. al. [8]), normally they are done as two separate stages.

Therefore a computer has difficulty reliably segmenting characters. For example, many groups take the approach of oversegmenting a word and then using a separate stage to combine these fragments into whole characters (e.g. Bozinovic and Srihari [2]). Whilst this two stage approach makes it easier for the computer to segment words, there is the possibility of error in both stages. If two segments which shouldn't be combined happen to look like two parts of one character, it is likely a segmentation error will result.

Following their success in speech recognition, Hidden Markov Models (HMMs) (see [37]) have been widely exploited in both segmentation-based and segmentation-

free systems. Several HMM methods for segmentation-based systems use letter sub-HMMs to handle the segmentation errors discussed above, e.g. [22]. Sub-HMMs are trained to model each letter in order to recognise them more reliably, or even combine the segmentation and recognition stages. A variant of these HMM methods uses a Variable Duration HMM (VDHMM) to cope with segments of different sizes [5].

Some non HMM approaches include [2] and [36]. Bozinovic and Srihari [2] segment each word into possible letter breaks, usually oversegmenting, and then extract features to distinguish between various letter hypotheses. Plessis et. al. [36] take a multi-classifier approach including post processing to overcome segmentation mistakes.

1.2.1.2 Segmentation-free Methods

In contrast, segmentation-free methods use features of the word image as a whole.

Govindaraju and Krishnamurthy [17] converted the word image into an ordered sequence of strokes. This recovered some of the temporal information that has been lost when using an off-line system. See Section 5.2 for more details.

Gorsky [16] used a holographic representation of the word. This means that each set of stroke fragments in approximately the same direction is mapped to a point in another parameter space. The new coordinates are made up of the number of fragments, the local order of the fragments, and the fragment direction. The intensity of each point represents the length of fragments sharing these characteristics.

Parisse [30] used the upper and lower profiles of the word as identification features. These profiles were produced by finding the outer contour of the word.

Some HMM-based segmentation-free methods include Guillevic and Suen [19] and Gilloux et. al. [15], both of which represent the word model as a change of identical sub-HMMs. Guillevic and Suen consider the possibility of skipping a state to allow for extra flexibility. They use a sliding window and low level features

such as slope of line fragments to produce a feature vector. In Gilloux et. al, each sub-HMM can have different internal transitions, allowing different paths through each sub-HMM to allow for character variability. Gillies [14] and Mohamed and Gader [28] use one pixel wide columns of the image as their observations.

1.2.2 Arabic Cursive text

In contrast to Latin text, much less work has been done on the recognition of Arabic. Since even printed Arabic text is in cursive form, segmentation of Arabic script into characters is more difficult.

This has perhaps led to a smaller proportion of systems that attempt segmentation before recognition. Of the systems that do, many segment not by character but by some other unit – parts which are easier to segment.

This means the line between segmentation-based and segmentation-free methods is a little more blurred. For instance, some methods segment into column-wide pixels and then use a HMM. These are treated here as segmentation-free methods since these columns are not really representative of any real segmentation in the word (and merely a convenient way to pass the image to a HMM), but it could be argued that they are actually segmentation-based.

1.2.2.1 Segmentation-based Methods

Gillies et. al. [13] constructed an entire unix-based Arabic text recognition system. Their approach was to over-segment the word to ensure that no segment belonged to more than one character. They then combined these segments in many possible ways and sent them to a trained neural network which recognised whole characters from among the options. These were passed to a Viterbi search (discussed in [37]) to perform sequence prediction for the word.

Elgammal and Ismail [9] segmented the words into ‘scripts’ (small connected segments), using a Line Adjacency Graph (LAG) [31], and then used features of

these scripts to classify them.

A popular approach has been to use a Hidden Markov Model to recognise the word using some sequence of word features. Dehghan et. al. [7] split words into overlapping vertical segments twice the width of the average stroke width. They then found column features, and reduced these using codebook vectors found by a Self-Organising Feature Map (SOFM). These were then passed to a HMM for sequencing.

An example where segmentation and recognition are performed simultaneously [8] scans the word accumulating one column at a time, until the segment is recognised a character. A segmentation point can then be inserted. This approach uses moment invariants [23] (see also Chapter 4) to recognise the characters, since these are independent of size, translation and rotation.

1.2.2.2 Segmentation-free Methods

This approach seems to be popular for Arabic text, and focuses on using features of the image as a whole to classify the word.

A wide range of different features have been used by different groups. Many of these features require normalisation of the image before features can be extracted. An important part of this is often finding the baseline of the word, i.e. the line on which it is written and by which the characters are often connected (see Chapter 2). Ways of doing this include horizontal projection histograms [7], word skeletons [32], and Hough Transforms [1].

Amin [1] used outer and inner contours of the word to detect loops and peaks in a word image, and passed these features to the C4.5 decision tree algorithm.

Erlandson et. al. [10] used the following whole-word features to create feature vectors:

1. Dots and similar small marks
2. Directional segments - stroke fragments with a certain orientation

3. Junctions and endpoints - crossing and end points in the word skeleton
4. Directional cavities - partially enclosed white space, similar to holes
5. Holes - the inside of loops in the word
6. Descenders (see Chapter 2) and intra-word gaps

which were then compared to a lexicon of training feature vectors to find the best match.

Pechwitz and Maergner [33] passed images to a HMM recognition system in columns one pixel wide. The HMM recogniser had sub-HMMs to recognise the characters.

1.2.3 Preprocessing

As well as the actual recognition stage, a handwriting recognition system must perform a variety of preprocessing tasks to ensure the text is in a suitable form. These steps include:

- Thresholding - distinguishing foreground pixels from background pixels in the handwriting image. Typically, a grayscale value is chosen, and any pixels darker than this are assigned to the foreground.
- Noise Removal - identifying pixels which represent noise and correcting them.
- Line and Word Segmentation - splitting the page image into areas representing lines of text and then words.
- Baseline Finding - finding the writing line, i.e. the line upon which the text sits (see Chapter 2).

Not all these may be necessary in each case. For more on preprocessing see for example [35] and [41]. The only one of these tasks we shall be concerned with in this project is that of finding the baseline, since the data we will be using

has already been preprocessed. The task of finding the baseline is explored in Chapter 2.

1.3 Database

The Arabic images used in this project are those in the freely available IFN/ENIT database of handwritten Tunisian town names [34]. This database was only recently assembled by Pechwitz et. al, and before this there was no standard database for this field. The database consists of a collection of town name images, each containing between one and four words. The images come from 411 different writers, mostly students, and consist of 946 different town names. All the figures in this thesis showing Arabic words are examples from this database.

Much of the necessary preprocessing has already been performed. The preprocessing stages already performed on the database images are (see Section 1.2.3 for details):

1. Thresholding
2. Noise reduction
3. Word segmentation

The work can therefore focus on ways to recognise these images without being slowed by the preprocessing, which can already be done accurately. The exception here is the task of finding the baseline, which will be covered in Chapter 2. The database includes data about the true baseline for each image, making it possible to evaluate baseline-finding methods.

Since the full database was very large, it was decided to produce a small dataset consisting of just 10 different town names, in order to obtain initial results. Performance over a certain value (e.g. 90%) on the small dataset would then warrant testing on the full database. This allows testing of many algorithms with less problems of time consumption and data storage.

1.4 Outline of Forthcoming Chapters

Chapter 2 describes work on finding the baseline of Arabic text images using Principal Components Analysis (PCA). Both a simple and improved method are described and compared with the performance of previous methods.

Chapter 3 investigates a simple Nearest Neighbour approach to classification using resized word images. The performance improvement gained by using *Tangent Features*, an analogue of Tangent Distance, is also investigated.

Chapter 4 covers methods using image moments for classification. A simple method exploiting Centroid information, is described along with a better-performing method using Zernike Moments as classification features.

Chapter 5 attempts to improve performance by retaining as much of the important information as possible at each stage of the recognition process. The path followed by the pen is inferred, and converted into a feature vector which can be used for classification. Also, the word image is split into smaller sections which are individually classified.

Chapter 6 discusses and summarises the work covered, its contribution to the field of Arabic handwriting recognition, and also suggests some possible future directions.

Chapter 2

Baseline

2.1 Introduction

Although many preprocessing tasks have already been performed on the database (see Section 1.3), the town names have not been corrected for word skew. This means that the imaginary line upon which the words are written may be rotated from horizontal.

This imaginary line is termed the *baseline*, and corresponds to the line upon which one would write on ruled paper. Arabic text is written along this baseline, with the majority of foreground pixels in this area. There are also lines sticking out above and below the baseline, and these are termed *ascenders* and *descenders* respectively. An example of a word with these features is shown in Figure 2.1.

Many text recognition methods use the baseline of the word in part of the recognition process. Finding the baseline is often an essential step before feature extraction can be performed. Pechwitz and Maergner [33] describe how improvements in baseline accuracy can make a large difference in performance for a system using the IFN/ENIT database.

Knowledge of the relative position of part of a word in relation to the baseline is also very important. Consider the recognition in Latin text of the numeral ‘9’

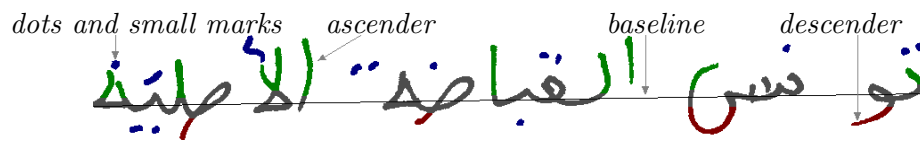


Figure 2.1: *Some features of Arabic words. Baseline (black), dots and other small marks (blue), ascenders (green), descenders (red).*

versus the letter ‘g’. Some writers will write these with the same shape, differing only in the vertical position relative to the baseline.

Since automated recognition systems often have problems with generalisation, the recognition of a rotated image can be problematic. In many schemes it therefore makes sense to use the baseline of a word to correct for its rotation.

2.2 Previous Work

Some methods of finding the baseline include:

- Projection Histogram - Here a projection profile of the image is built up by taking horizontal slices across the image, and summing the number of foreground pixels. The position of the baseline tends to be indicated by a peak in the distribution (see Figure 2.2). This is because the majority of pixels lie along the baseline. By performing this several times with slightly different rotations of the word, the largest peak can be found. This peak indicates both the position and orientation of the baseline. Projection histograms work best on long words and often give erroneous predictions for short isolated words. An example of this method is work done by Kanai and Bagdanov [24] to find page skew.
- The Hough Transform - Each point in Hough space corresponds to a line at a certain angle and distance from the origin. The value at this point is an expression of how many datapoints lie on that line. The Hough Transform can be used as a way of finding straight lines in a set of data. Points in Hough space with a high value signify many datapoints lying along the

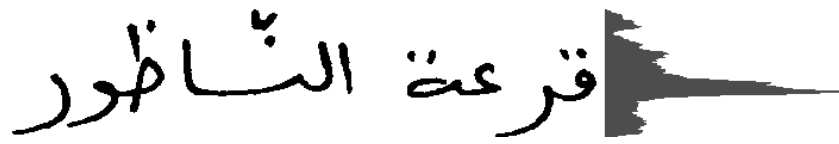


Figure 2.2: *Projection histogram to find baseline position. The vertical position of the baseline corresponds to the largest peak of the histogram.*

corresponding line in Cartesian space. This can be used to find the baseline because it is characterised by many foreground pixels lying along a straight line. Yu and Jain [43] used the Hough Transform to detect page skew.

- **Skeleton-based Methods** - Produce a polygonal approximation to the skeleton of the word. Specific features of this skeleton are then used to determine the baseline. This was done for Arabic text by Pechwitz and Maergner [32]. Whilst this method works well, each set of features can only be used for a specific type of text.
- **Nearest Neighbour Approach** - A connected component analysis is first performed on the image (groups of connected foreground pixels found). Each connected component is then linked to its nearest neighbour by drawing a line connecting the centre of mass of each. By averaging the angles of these links, an estimate of baseline angle can be calculated. This Nearest Neighbour approach was used on document images by Hashizume et. al. [21]. On isolated town names however, local variations in ascenders, descenders etc. are likely to make this method inaccurate. It should be stressed that this Nearest Neighbour approach refers merely to the fact we connect each component with the one nearest to it. This should not be confused with the usual use of this term in classification systems (see Chapter 3).
- **Principal Components Analysis (PCA)** - Is used to find the principal axis of the foreground distribution (see Section 2.3). This gives an angle for the baseline but not necessarily its vertical position. Use of this method on Latin text is investigated by Steinherz et. al. [40]. They found a higher performance when using the background pixels for PCA rather than the foreground.

2.3 Principal Components Analysis

Principal Components Analysis is a way of finding the directions along which a distribution exhibits the greatest variation. These directions are termed the Principal Components of the distribution. They correspond to the most significant eigenvectors of the covariance matrix of the datapoints.

If the datapoints are the coordinates of foreground pixels in a town name image, the most significant of these eigenvectors describes the direction of the baseline. As mentioned in Section 2.1, Arabic words are written lying on a baseline, with ascending and descending strokes. The word will therefore be distributed around the central axis of the baseline. It is the angle of this central axis which is given by PCA.

This approach makes the assumption that the foreground pixels of the image are evenly distributed around the baseline. It should be noted that this assumption is not valid for all words, as we shall see in Section 2.4.

Pechwitz and Maergner have explored the performance of some common baseline finding methods on the IFN/ENIT database in [33]. However, the author could not find an example of PCA being used on this database. It was therefore decided to investigate the performance of this PCA method on the IFN/ENIT database.

2.3.1 Implementation

MATLAB code was written to perform PCA on an image in order to find its principal axis, and thus give an indication of the baseline. This only gives an angle for the baseline however. The vertical position was determined by rotating the image by this angle and finding the peak of its projection histogram. The code has the following steps:

1. Convert the image into a set of vectors describing each foreground pixel that makes up the town name.

<i>Method</i>	<i>Percentage acceptable (≤ 7 pixels)</i>
PCA (foreground)	82%
PCA (background)	81%
Hough Projection [33]	83%
Skeleton-based [33]	88%

Table 2.1: *Performance of PCA baseline estimation vs. other methods.*

2. Perform PCA on these points and pick the eigenvector with the largest eigenvalue. This vector gives the direction of the baseline.
3. Images are rotated so that this estimated baseline angle lies horizontal. A projection histogram is then taken of the image.
4. The peak of this projection histogram is used to determine the vertical position of the baseline.

As mentioned in Section 2.2, Steinherz et. al. [40] showed that accuracy was increased by using the background rather than the foreground pixels as points for the PCA algorithm. Therefore, an experiment was also performed using the background pixels for PCA.

2.3.2 Results

Experiments were performed using the foreground or background pixels for PCA. The results were compared with the results of Pechwitz and Maergner [33] on the IFN/ENIT database. They used the percentage of examples with an acceptable baseline estimate as a performance measure. They define an acceptable estimate as one where the average distance between the true and estimated baseline is ≤ 7 pixels.

Table 2.1 shows the results of PCA compared to the results of Pechwitz and Maergner for Hough Projection and Skeleton-based methods. Due to time constraints, the first 1000 town names in the database were used for the PCA results,

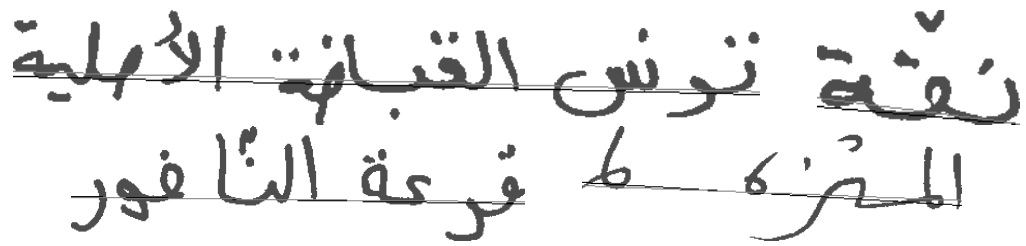


Figure 2.3: *Some example results of the basic PCA method. The darker line is the true baseline, the pale line is the estimated baseline.*

rather than the full database. Since this test set was slightly different to the one used by [33] (being a subset of this), fractions of a percent were considered not significant, and so all percentages were rounded for comparison.

A selection of the results was also inspected visually. Some examples are shown in Figure 2.3.

It should be noted that on some words however, the estimated angle was disrupted by the uneven distribution of dots, ascenders and descenders. This problem is discussed, along with examples, in Section 2.4.

2.4 Improved PCA

The above method worked well for most cases, but the baseline of some shorter words was adversely affected by ascenders and descenders. These are lines sticking out above and below the baseline. These ascenders and descenders alter the distribution of the pixel coordinates in such a way that the principal axis is rotated. This is especially pronounced when a short word starts with a descender and ends with an ascender or vice versa. In this case one end of the estimated baseline will be pulled down and the other pulled up. This will result in a less accurate baseline.

The baseline may also be affected by the dots and other similar marks (e.g. diacritics) in the image. These dots are often quite small in area, but tend to lie a long way from the true baseline. They can therefore cause the central axis



Figure 2.4: *The influence of word features on the baseline. The darker line is the true baseline, the pale line is the estimated baseline.*

determined by PCA to shift significantly.

Examples of town names affected by these points are shown in Figure 2.4

2.4.1 Implementation

In order to overcome these limitations, some way of reducing the influence of the ascenders, descenders, and dots is required. The problem of ascenders and descenders was reduced by skeletonising the word and then removing pixels that lie at the end of a stroke. These will be pixels that have only one other foreground pixel in their 8-neighbourhood. This corresponds to spur removal and can be performed by the MATLAB function for this task.

By repeating this removal of pixels, line strokes can be progressively shortened from the ends inward. Since ascenders and descenders are usually formed from the start or end of a line stroke, these will be removed at an early stage of the process. By selecting a suitable number of spur-removal iterations, the influence of these ascenders and descenders can be reduced. By visual inspection, a value of 10 iterations was chosen as suitable.

Removal of dots and similar marks is performed by finding areas of foreground pixels less than a certain value, and removing these with MATLAB's `bwareaopen()` function. By visual inspection of these marks, a value of 300 pixels was found to be suitable.

The steps followed by the improved code are:

1. Take the image and skeletonise it .

<i>Method</i>	<i>Percentage acceptable (≤ 7 pixels)</i>
PCA	82%
PCA with dot-removal (≤ 300)	82%
PCA with dot-removal (≤ 300) and spurring (10 iterations)	79%

Table 2.2: *Performance of PCA improvements.*

2. Remove the smaller connected components in order to remove the influence of dots and other small marks.
3. Perform spur removal a certain number of times, determined by visually inspecting a small set of example images.
4. Convert the background pixels into a set of vectors describing these points.
5. Perform PCA on these points and pick the eigenvector with the largest eigenvalue. This vector gives the direction of the baseline.
6. Images are rotated so that this estimated baseline angle lies horizontal. A projection histogram is then taken of the image.
7. The peak of this projection histogram is used to determine the vertical position of the baseline.

2.4.2 Results

Since using the foreground pixels gave the best results in Section 2.3.2, they were used for all the results in this section. Table 2.2 shows the results of dot removal (anything ≤ 300 pixels), and dot removal followed by spurring (10 iterations). The result using the original image is shown again for comparison.

Figure 2.5 shows two examples where an improvement in performance was seen. In the first example, we can see how the small marks above the word caused the baseline angle to be too steep. This is corrected by the removal of these marks

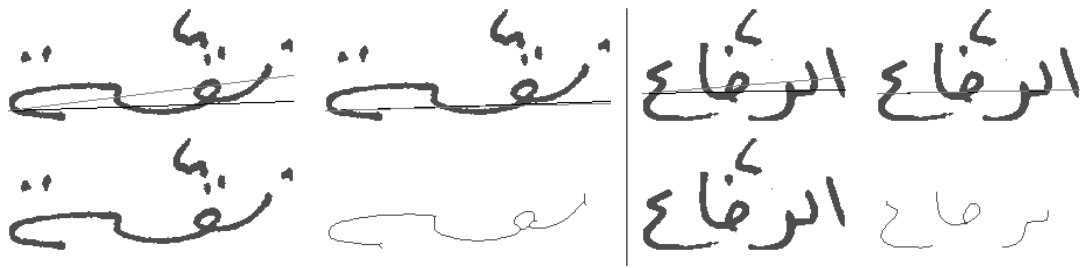


Figure 2.5: Examples of improved PCA performance. Top row - baseline results, bottom row - image used for PCA. The darker line is the true baseline, the pale line is the estimated baseline.

before PCA is performed. In the second, a combination of dots and ascenders and descenders act to skew the baseline estimate. The dots are removed and the influence of the ascenders and descenders is reduced using spur-removal. However, there was no significant performance increase in the results as a whole.

2.5 Discussion

The results in Section 2.4.2 show that PCA compares favourably with the other methods investigated by Pechwitz and Maergner [33]. Performance is very similar to the Hough Projection method, and slightly less than that of the Skeleton-based method.

Since finding the baseline is a critical step in many recognition tasks, another way to compare PCA to other methods would be to use it as part of a larger system. For example Pechwitz and Maergner [33] constructed a system using a HMM for recognition, and showed the results were dependent on baseline accuracy. A similar system could be run multiple times using different baseline methods, and performance compared.

Using the foreground pixels gave 1% higher performance than using the background pixels. Why this is the case when Steinherz [40] found the background pixels worked better is unknown. Since the difference is small, it could just be a property of the database used. It is possible that the pixel distribution of some

town names is such that the principal axis actually lies at a right angle to the baseline. In this case the baseline would correspond to the second most significant eigenvector. This is likely to be one of the sources of error in the performance of the method as a whole, but it may happen with slightly different frequency when using the background pixels.

To avoid this problem of the estimated baseline being at right angles to the true baseline, the algorithm could be adjusted. This would involve considering the two most significant eigenvectors, rather than just one. One of these will be the correct baseline estimate. The algorithm now needs to decide between these two possibilities in an intelligent way. One effective way to do this would be to project both onto the image, and choose the line which has the most foreground pixels lying on it. The correct baseline eigenvector will have many foreground pixels, whereas the other will be at right angles to the town name, and will therefore have very few foreground pixels.

It was surprising that despite improving performance on individual test town names, the improvements suggested in Section 2.4 did not increase performance overall. In the case of dot removal, this could be that most of the town names where a significant improvement was shown already had an estimate lying within seven pixels of the baseline. There was therefore no significant performance increase as a whole.

Spurring before PCA actually caused a slight decrease in performance. The main assumption of this spurring idea was that ascenders and descenders often lie at the end of a pen stroke, so removing the ends of these lines would partially remove them. In reality however, many strokes lie either mostly above or below the baseline, and spurring causes the stroke to shrink toward its own centre. If this centre is a significant distance from the baseline, this can cause a decrease in performance, as observed.

Chapter 3

Nearest Neighbour & Tangent Features

3.1 Introduction

Since the problem of Arabic text recognition is a large and complex one, it makes sense to first try a simple method to see what performance can be achieved. The performance of more complex methods can then be compared to this.

It was hard to know at this stage what features of the images would be most suitable for classification. In order to retain as much information as possible it was decided to compare the images in their raw form, i.e. an ordered set of pixels.

This chapter first considers a basic Nearest Neighbour method of classification (Section 3.2). An improvement to this using *Tangent Features*, an analogue of Tangent Distance, is then explored (Section 3.3).

3.2 Nearest Neighbour

One of the simplest classifiers we can use is the Nearest Neighbour classifier. This takes a test point in vector form, and finds the Euclidean distance between this and the vector representation of each training example. The training example closest to the test point is termed its Nearest Neighbour. Since this example is in some sense the one most similar to our test point, it makes sense to allocate its class label to the test point. This exploits the ‘smoothness’ assumption that points near each other are likely to have the same class.

3.2.1 Implementation

One of the possible limitations of the Nearest Neighbour method is that all the data points must be of the same dimension for the comparison to make sense. Since the images in the database are of varying sizes, they were all resized to some standard size. Each image was then formed into a single column, which made up the feature vector for the Nearest Neighbour classifier. Whilst the database images are binary, the resized images were kept as grayscale, as this retains more of the neighbourhood information for each pixel in this resized image.

The only parameter of our classifier to still be decided upon was the dimension of these resized images. Due to the town names having varying lengths, even the image proportions vary. To give some indication of the most sensible proportions, the ratio of height versus width for each image was calculated, and an average taken. This suggested that a suitable image proportion would be to have images with a width four times their height.

To decide on the actual size of the images, it was important to be aware of the trade-off between classification rate and losing information. The larger the images are, the more storage space they will take up and the slower the algorithm will run. Indeed, the classification rate will decrease exponentially with the size of our feature vector (the so-called ‘Curse of Dimensionality’). Conversely, making



Figure 3.1: The town name from Figure 1.1 resized to 32×128 pixels. It is still readable despite being resized.

the images too small will mean too much information is lost.

It was decided to set the image size visually by resizing the image and then verifying it was still readable. A size of 32×128 pixels was chosen as suitable (see Figure 3.1). Whilst this seems a reasonable trade off between the different factors, the feature vector produced was still 4096 dimensional. Since this took a long time to classify, an image size of 16×64 pixels was also tried, resulting in a 1024 dimensional feature vector.

3.2.2 Results

Performance on the small test set was measured using a simple accuracy figure. This describes the percentage of test examples that were correctly classified. The results are summarised in Table 3.1.

<i>Image Size (pixels)</i>	<i>Accuracy</i>
32×128	78%
16×64	81%

Table 3.1: Performance of simple Nearest Neighbour models on the small data set, using resized images.

3.3 Adding Tangent Features

It was then decided to try to improve performance by using an analogue of Tangent Distance, termed *Tangent Features*.

Tangent Distance is often used as an alternative distance metric to the standard Euclidean distance. Whereas Euclidean distance can be thought of as measuring the distance between two points, Tangent Distance can be visualised as finding the shortest distance between two planes. Each of these planes can be considered to be an approximation to a manifold in the feature space. This manifold is formed by transformation of the test point, moving it through feature space.

The transformations to be approximated are chosen so as to reflect the variability of the datapoints. For example we might consider a set of words to be classified where the angle at which the words are written varies. In this instance the primary variation is this rotation of the words, so the transformation in feature space is chosen to reflect this rotation.

Tangent Distance has been used for many similar handwriting recognition tasks, including handwritten digit recognition as described by [38]. This paper also gives a more detailed explanation of Tangent Distance along with mathematical details.

Since calculation of the true tangents of some non-linear transformation functions is computationally intensive, an approach was taken to approximate these tangent planes by generating extra datapoints. We do not know what the most important transformations are for this dataset so it makes sense to determine these from the data. Subtracting one image from another in the same class results in a matrix describing the transformation between them. However, performing this for each datapoint in the class, and then adding these transformations to the datapoints, will simply result in the original dataset. A way needs to be found to extract the most significant transformations on average per class.

Also, if we were to pick five significant transformations and apply them to each

datapoint, a dataset five times as large would result. Whilst this may improve accuracy, there will be a big drop in performance of the Nearest Neighbour algorithm.

3.3.1 Implementation

It was decided to find the mean of each class and apply a certain number of significant transformations to form a smaller dataset. To find the most significant transformations in each class, the mean was first subtracted from each datapoint to produce a new set of points. PCA was then used to pick the N most significant eigenvectors. These should hopefully describe the N most significant transformations.

Code was written in MATLAB to perform the following steps:

1. Resize each image and convert to a feature vector as described in Section 3.2.1.
2. Form these feature vectors into matrices where each column corresponds to one datapoint.
3. Find the mean of each class in the training set and subtract this from each of the training set datapoints in that class.
4. We now have matrices describing the transformations away from the mean for each datapoint in a class.
5. PCA is performed on these matrices and the five most significant eigenvectors are returned.
6. A new training matrix is then formed in which each class is represented by six datapoints, the first of which is the mean of the class, and the others are generated by applying each eigenvector to the mean.
7. Nearest Neighbour classification is then performed as before.

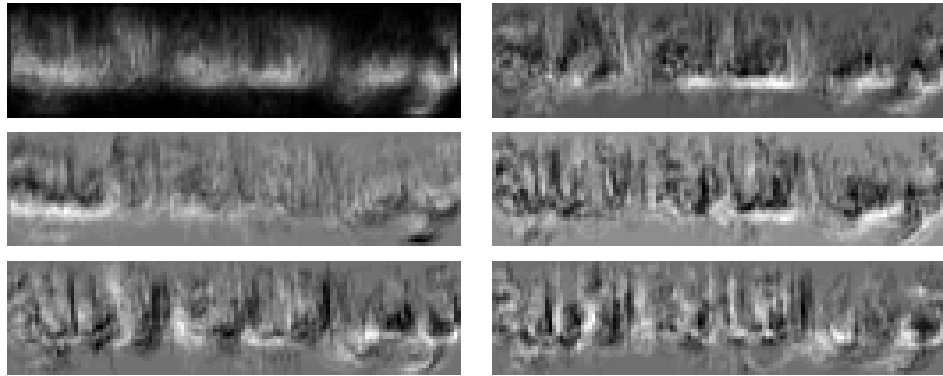


Figure 3.2: *Mean (top left) and first five eigenvalues of the transformations within one class of town name.*

3.3.2 Results

The mean and eigenvectors of resized images are shown in Figure 3.2 for one class of town name.

This Tangent Feature method was applied to the best performing image size from Section 3.2, i.e. 16×64 pixels. This resulted in a classification accuracy of 94% for the small test set. A training set consisting of just the means of each class was also tried for comparison. Surprisingly, this resulted in a classification accuracy of 90%. Since the results were so encouraging, the same method was applied to the full database. However, here performance was much lower at 52%, or 50% using just the means.

3.4 Discussion

3.4.1 Nearest Neighbour

It can be seen that the smaller of the two image sizes performed slightly better. This is presumably because the lower resolution affords better generalisation capabilities. Also notable is the fact that the performance of this very simple method is reasonably high, at 81% on the small dataset.

It is often the case in this type of complex problem that a simple Nearest Neighbour method performs well. This is because the instance-based nature of the algorithm allows it to cope with complex distributions of the classes. It is not a very good method for generalising however, as can be seen in the performance increase of just using the means.

3.4.2 Tangent Features

Performance using Tangent Features was encouragingly high on the small test set. More surprising was the performance using just the means of each class (Section 3.3.2). Presumably this still high performance comes from the generalisation capabilities of just using the means.

The high performance encouraged testing the Tangent Features method on the full database, but here performance was much lower. The larger number of classes means that there will be many town names with approximately the same shape. This results in the means of many classes being very similar and so raising the chance of misclassification.

It would be interesting to compare the performance of our Tangent Features approach to the more complicated Tangent Distance method. Whilst the transformations used when calculating Tangent Distance are usually selected by hand, the advantage of the Tangent Features is that these transformations are inferred from the data.

Chapter 4

Centroid & Moments

4.1 Introduction

We now consider one of the major factors causing performance degradation when simply using resized images in a Nearest Neighbour classifier. Slight variations (e.g. a small rotation) can change a lot of pixels, and thus create a large shift in Euclidean Distance.

Section 3.3 described an attempt to reduce this problem by generating extra datapoints to represent common transformations. It might be better to find features of the word that are invariant to some of these transformations, and use these as a feature vector. This approach is the subject of this chapter. Specifically, this chapter looks at the Centroid and Moments of an image.

The Centroid can be thought of as being the centre of mass of an image. For a town name, if each of the foreground pixels is considered to have an equal mass, we can find the Centroid by calculating the position of the centre of mass of this system.

Moments of an image can be thought of as the decomposition of the image into a series of numbers that describe the distribution of the image function. This is similar to a cloud of datapoints being split into a set of components through

PCA, or the decomposition of a waveform into its frequency components by a Fourier Transform.

4.2 Previous Work

The original work on moment invariants was done by Hu [23]. He derived equations for seven quantities invariant to rotation, translation and scaling. These quantities were calculated by combining moments of the image.

Since then there has been a lot of work on both deriving different types of invariants, and using invariant moments for image recognition. Some examples include:

- In addition to deriving moment invariants, Hu [23] gives an example of their use in the recognition of Latin characters, but concluded more invariants were needed to achieve acceptable performance.
- Mamistvalov made some corrections to Hu's work [26], and also generalised this work to the n -dimensional case [27].
- Zernike [44] introduced a set of complex polynomials that formed a complete orthogonal set over the interior of a unit circle. Teague [42] then introduced the idea of projecting the image function onto orthogonal polynomials. Teague also described the projection of the image function onto the Zernike basis set mentioned above to produce Zernike moments. These are invariant with respect to rotation. In comparison, Hu moments are not orthogonal, so there is redundancy in the information they capture.
- Flusser and Suk [11] derived a new set of moment invariants that were invariant under affine (linear) transformations. They can therefore handle character and word slant, but not non-linear distortion such as rotation. They described an application of this work in [12], and showed that their Affine moments outperform Hu moments when recognising linearly distorted Latin characters.

- Khotanzad and Hong [25] give some example applications of Zernike Moments to image recognition. They describe the recognition of both Latin characters and lake outlines. This paper also details their method of compensating for translation and scaling of the image as well as rotation.
- Chong et. al. [6] derived translational invariants of Zernike Moments, and then described experiments on Latin character recognition. However, this involved calculating shifting factors, which introduces complications not suffered by Khotanzad and Hong's method of simply moving the origin to the image centroid.
- El-Dabi et. al. [8] present a recognition system for typewritten Arabic text. This system uses invariant moments to recognise the Arabic characters in a word. Pixel-wide columns of the image are accumulated until the moments of this image portion are found to be similar to an example character. The image portion is then allocated to that character, and the process starts again on the next image portion. In this way the characters are recognised and segmented simultaneously.

4.3 Radial Method

As a first attempt to extract invariant features, the following scheme was devised. First the origin of the coordinate system is moved to the position of the image Centroid. This should solve the problem of translation of the image, i.e. shifts in the whole image position. Coordinates of each pixel were transformed from a Cartesian to Polar representation and then the angular coordinate was discarded. This should solve the problem of rotation of the image, since only the distance of each pixel from the origin is retained. These distances were then scaled to lie in the range 0 to 1 to normalise them. This should take care of the scale issue. Figure 4.1 gives a demonstration of these points.

A way to visualise this would be to have each pixel send out a signal which travels



Figure 4.1: Example of the Radial Method using 10 bins. The number of pixels falling in each section form a 10-dimensional feature vector.

at a constant rate toward the Centroid. We have a receiver at the Centroid, and record a trace over time of the intensity of the incoming signal. This intensity corresponds to the number of pixels sending a signal which arrives at one particular time.

This is similar to a suggestion of how the V2 area of the visual cortex works [18]. Because of the radial nature of this scheme, it will be termed the *Radial Method*.

4.3.1 Implementation

This Radial Method was implemented in MATLAB by the following steps:

1. The Centroid coordinates of each binary image are calculated.
2. The coordinates of each foreground pixel are then found.
3. The distance of each foreground pixel from the Centroid is calculated.
4. Each distance is divided by the largest distance value to normalise it.
5. The number of pixels lying within a certain distance range are calculated, and the value put in a bin corresponding to that range.
6. There will be M bins of equal width lying in the range 0 to 1. This produces an M -dimensional feature vector containing bin values.
7. Nearest Neighbour classification is performed on these feature vectors.

4.3.2 Results

The Radial Method gave quite poor performance on the small test set. The results for different numbers of bins are shown in Table 4.1. The reasons for this poor performance are discussed in Section 4.5, but it was decided that the main reason was that too much information was being lost. This motivated the use of image moments (Section 4.4).

<i>Number of bins</i>	<i>Accuracy</i>
10	50%
20	51%
50	52%
100	49%

Table 4.1: *Performance of the Radial Method on the small testset, using different numbers of bins.*

4.4 Zernike Moments

As mentioned in Section 4.2, invariant moments have been used for various pattern recognition tasks with some success. It was therefore decided to try them on the problem of recognising the images in the IFN/ENIT database. The original moment invariants are described in [23] and are termed Hu moments. These are invariant to translation, rotation and scaling. However, there are only seven of these, which are unlikely to incorporate enough information about the town names. Indeed, initial experiments using these moments yielded poor performance of only 35% accuracy on the small dataset.

Another type of moments, called Zernike Moments, can be calculated to whatever order is desired, as there is a general formula describing them (see Equation 4.1 for the version for a binary image). The order of a Zernike moment is determined by the value of n in this equation. When we calculate all Zernike moments up to

order N , we use all the Zernike moments where $n \leq N$.

$$Z_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x, y) R_{nm}(\rho) \exp(-im\theta) \quad (4.1)$$

where

$$R_{nm}(\rho) = \sum_{s=0}^{n-|m|/2} (-1)^s \frac{(n-s)!}{s! \left(\frac{n+|m|}{2} - s\right)! \left(\frac{n-|m|}{2} - s\right)!} \rho^{n-2s} \quad (4.2)$$

and

- n Positive integer or zero.
- m Integers subject to constraints $n - |m|$ even, $|m| \leq n$.
- ρ Length of vector from origin to (x, y) pixel.
- θ Angle between vector ρ and x axis in counterclockwise direction.

Since an arbitrary number of Zernike Moments can be calculated, this makes them a lot more suitable for our purposes. This higher number of moments should give better performance.

One drawback to using Zernike Moments however, is that they are only invariant with respect to rotation. Invariance to translation and scaling of the town name image is also desirable. It was therefore necessary to move the origin of each image to its Centroid, and to scale the image by the area of its foreground pixels before the moment calculation.

4.4.1 Implementation

The MATLAB algorithm for this method has the following steps:

1. Move the origin of each image to its Centroid.
2. Resize each image to standardise the area of its foreground pixels. Each resized image is left in its grayscale form.
3. Zernike moments are then calculated up to order N as described by Equation 4.1.

4. These moments are then used as the feature vector for a standard Nearest Neighbour classifier.

4.4.2 Results

The results obtained on the small testset using different orders of Zernike moments are summarised in Table 4.2. A much higher number of moments was also tried, but resulted in much lower performance. This is presumably due to the extra detail spoiling the generalisation abilities of the classifier.

<i>N</i> - order of moments	Number of moments	Accuracy
2	4	48%
5	12	58%
9	30	74%
12	49	76%
20	121	80%

Table 4.2: Performance on the small testset of Zernike moments of various orders.

4.5 Discussion

The performance of the Radial method described in Section 4.3 was surprisingly poor, which can be explained by two points. Firstly a slight change in part of a word will change the position of many pixels in relation to the Centroid, and may cause them to ‘hop’ from one bin to its neighbour. Secondly, a lot of position information is being lost when we throw away the angle coordinates of each pixel. The information retained by the Radial method may be sufficient to recognise simple shapes, but not complex word images.

Using the first 121 Zernike moments resulted in a classification accuracy approximately equal to the best performance quoted in Section 3.2. However this performance was achieved with feature vectors approximately one eighth the size of the resized images. This allowed a large increase in speed of the algorithm, and a reduction in its memory requirements.

The remaining 20% of test examples that were misclassified can be explained by more local variations in the words. Whilst this Zernike moments method is invariant to global rotation, translation and scaling, it is not invariant to variation in parts of this town name. For instance, two examples of the word 'dog' which are identical apart from the letter 'g' having a longer tail, will have different moments. There is also the potential for letter and word skew causing change in the moments. A lot of the variations in handwritten words are of this more local nature, which is hard to deal with when looking at the town name image as a whole.

Chapter 5

Tracing Approach

5.1 Introduction

So far we have looked at a variety of methods which rely on features of the word image as a whole. Whilst this is a viable approach when we have a small lexicon (such as the 10-class small dataset), we have seen how it can be problematic when our lexicon is large (Section 3.3.2).

The methods discussed in this chapter are motivated by two major considerations. The first of these is the fact that whole-word features have trouble with the lexicon size of the full database. It was decided that a better system would first attempt to recognise parts of the image before trying to classify the full town name.

As mentioned in Section 1.2.2, segmentation into characters is difficult for Arabic cursive text. However, Arabic words are often not fully connected, and are made up of more than one connected component. It is these components, hitherto termed Part Of Words (POWs), which will be recognised. In contrast to individual characters, these POWs are easily separated since each is surrounded by white space (see Figure 5.1). A typical town name will have several of these components and each can be recognised individually before an overall classification is decided upon. It is hoped that this method will be better able to cope

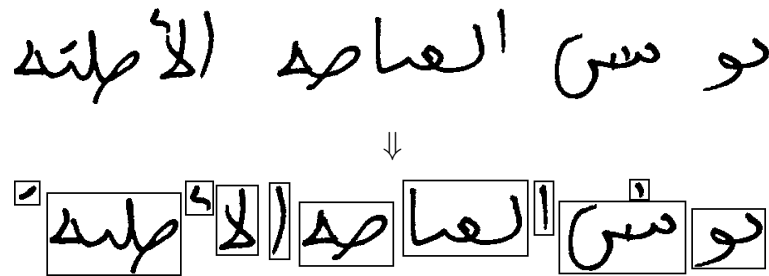


Figure 5.1: *Splitting a town name into POWs.*

with the large lexicon of the full database.

The second consideration is that we are not interested in the whole image. All the background pixels, and those depending on line thickness, are not really relevant to the recognition task. Also, the internal representation as images takes up much more space than an on-line representation would. These considerations motivate an attempt to trace the lines making up the town name, and to use these as a representation. Since we will be tracing the lines formed by the pen, some temporal information will be recovered. We will in effect be converting an off-line representation into a pseudo-on-line representation. This means that we can focus on the strokes making up the image – the only parts we are really interested in.

Because dots are effectively points, it makes little sense to trace them. The dots are therefore removed before segmenting the town name into POWs (as in Figure 5.1).

Another point to observe is that on-line recognition tends to have a higher accuracy than its off-line equivalent [35]. We might hope that this method could take advantage of some of this extra performance.

5.2 Previous Work

There seems to have been little interest in this specific area in the past. However, a few groups have looked at the reconstruction of on-line information from off-line

images with a view to recognising them.

Govindaraju and Krishnamurthy [17] traced word strokes, and then used the up and down strokes that were detected as classification features. Specifically, they used the following stroke features:

1. Orientation (Up or Down)
2. Slope of stroke
3. Length of stroke
4. Stroke start and end points

and then compared the feature vectors to precomputed vectors in a lexicon.

Nishida [29] attempted to recreate on-line data from off-line images and then use on-line recognition algorithms to classify words.

5.3 Initial Method

5.3.1 Tracing

The first stage in the process is to work out how each part of word has been formed by following the line which made it.

MATLAB code was written to perform the function of tracing this line. This was accomplished by finding the right-most end point of the line and following it to the other end. Once the first end point had been found, pixels lying in a circle of a certain radius around this point were investigated to determine where the line went next. This radius determines the step size of the trace, analogous to the sampling interval for genuine on-line data. A new point is chosen from the foreground pixels lying on this circle, and the process is repeated.

When there is more than one candidate pixel for the next step, some method of choosing between them was needed. In the event of a choice, the pixel which

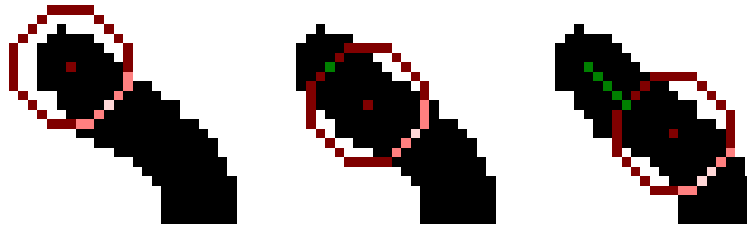


Figure 5.2: *The basic tracing process. Once the start point is found, a circle is drawn and foreground pixels lying on this circle (paler pixels) are found. Pixels too near the edges of the pen line are discounted, leaving candidate pixels for the next trace point (even paler pixels)*

requires the smallest change in line direction is chosen. This is because pen strokes tend to have smooth changes in direction, so it makes sense to impose this ‘smoothness criterion’. Also, pixels considered to lie too close to the edges of the pen line are discounted to avoid the trace following the outer contours.

Figure 5.2 shows three steps in the tracing process. A circle is drawn around the starting point, in this case of radius 6 pixels. The paler pixels show where the circle lies on foreground pixels. Once pixels lying too near the edge of the pen line are discounted, we are left with the (even paler) pixel in the middle of the pen line. This is chosen as the next point on the trace, and the process is repeated.

In terms of the tracing algorithm, the pixel satisfying the smoothness criterion is the one lying most directly opposite the penultimate point on the traced line. the rightmost picture in Figure 5.2 shows such an instance. Here, the two palest pixels on the circle represent the candidates, and the so-called penultimate pixel is the one which terminates the line on the other side of the circle. In this case, the top right candidate will be chosen for the next step.

This also allows the algorithm to deal intelligently with intersection points. An example is shown in Figure 5.3. It can be seen that the trace takes the correct path because of this smoothness criterion.

Another problem was the algorithm picking a path it had already traversed at one of these intersection points. This was remedied by removing the line from

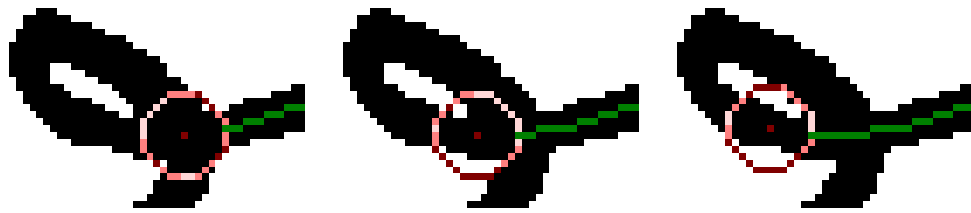


Figure 5.3: *Tracing at junctions. Here the algorithm takes the correct path by finding the candidate pixel that causes the least change in direction of the tracing line.*

the image as it was traversed, in effect erasing the line as we follow it.

When erasing the line in this way, there is the possibility that the tracer will reach a point where there are no foreground pixels lying on its circle of interest. This could mean it has reached the other end of the line, but there is the possibility that there is still more of the stroke to trace. When this circumstance arises, the algorithm searches for remaining foreground pixels which belong to the current POW, and picks the nearest of these as its new trace point.

In this way each POW can be transformed from an image segment into an ordered series of points describing the trace of its constituent pen line. This data is analogous to the time-ordered series of points generated by an on-line handwriting capture system.

5.3.2 Feature Extraction

The next stage was to turn the ordered series of points from the tracing algorithm into a concise feature vector. This can then be used to identify the class of the test POW by comparison to the POWs in the training set.

One way to do this is to look at the series of vectors linking each point to the next in the sequence. This can be visualised as a chain of arrows following the path the pen took. The angle between each of these vectors and its predecessor can then be used to form a sequence of numbers, which can be used as a feature vector. This series of angles is a compact way to describe the shape of the trace.

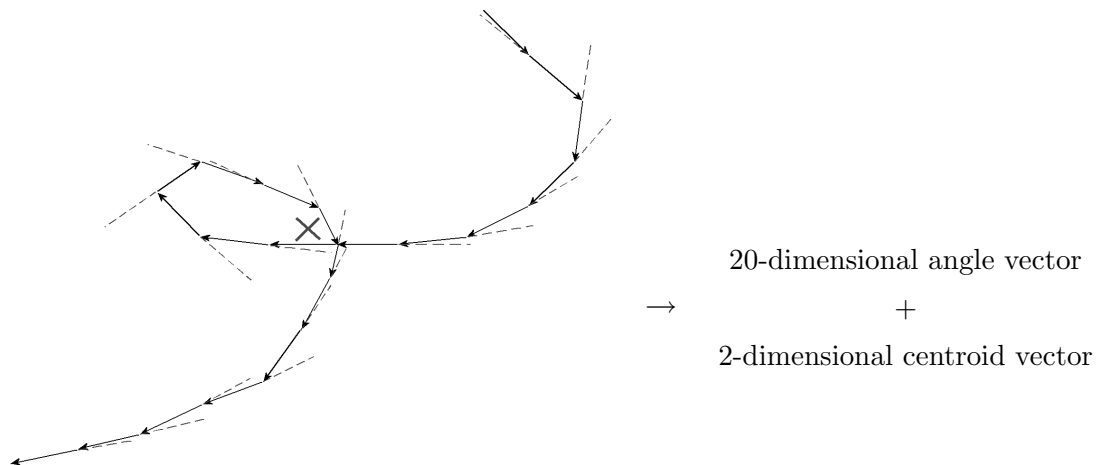


Figure 5.4: *Feature vector formation. The X marks the POW Centroid, and the dotted lines give an indication of the angle between each trace segment.*

Because each POW may be of a different length however, the number of points may vary. This would produce feature vectors of different sizes. Since these cannot be directly compared, the number of points is normalised to some standard value by interpolation. The feature vectors formed will therefore all be of a standard length, and thus are directly comparable. By visual inspection of the interpolated points, 20 was chosen as the standard length. This was found to be enough to describe a recognisable POW.

The first method tried uses this technique to form a feature vector of angles for each POW. The Centroid coordinates of this POW are also noted. Figure 5.4 shows the conversion for one POW. This is done for each POW in the training set of images. Each test image is then subjected to the same procedure and converted into several POW feature vectors and Centroid coordinates. The POWs are then individually classified.



Figure 5.5: *Confusion matrix for the initial tracing method. True labels run down the left hand side, and predicted labels along the top.*

5.3.3 Classification

The Euclidean Distance between the angle feature vector for the test POW and each training POW angle vector is found. Similarly, the Centroid coordinates for the test POW are used to find the Euclidean Distance to the Centroid of each POW in the database. The corresponding elements of the two resulting lists are then multiplied together. This results in a list of values, one for each POW in the training set. These values are an expression of how similar our test POW is to each of those in the training set, lower values being more similar. The class of the most similar POW is then allocated to our test POW as in a standard Nearest Neighbour scheme.

Once this has been done for each POW in the test image, a majority vote is taken on its overall class. This is repeated for each image in the test set.

5.3.4 Results

Performance of this initial tracing method was disappointing, at 47% accuracy on the small testset. Figure 5.5 shows the confusion matrix for this result, on the ten-class problem of the small testset. It can be seen that the top-right of the matrix is sparser than the bottom-left.

5.4 Improved Method

One of the main problems with the above method is that the simple majority vote is prone to errors. This is partly because the `mode()` function used picks the label with the smallest value in the event of a tie. This explains the fact that the bottom-left of the confusion matrix in Figure 5.5 is more full than the top-right.

To avoid this problem, it would be good to assign ‘pseudo-probabilities’ to the potential POW labels, so that a draw at the majority vote stage will be highly unlikely, as the vote can be weighted by this pseudo-probability.

Since we are taking a pseudo-probabilistic approach, it is possible to further weight the vote by probabilities derived from whole word features. This will in effect act to filter out the potential classes that are unlikely, and hopefully increase performance.

Also, it was realised that too much information was still being lost. Therefore an extra POW feature was added, namely the length of the line making up the POW.

5.4.1 Tracing and Feature Extraction

Each town name image is split into POWs as before, and each POW is traced as described in Section 5.3.1.

The trace for each POW is turned into three feature vectors:

- The series of angles between the tracing steps, as detailed in Section 5.3.2, of dimensionality twenty.
- The Centroid of the POW, also described in Section 5.3.2, is two-dimensional.
- A one-dimensional number expressing the length of the line which has been traced by the algorithm. This length is divided by the horizontal size of the image, in order to obtain a proportional length between 0 and 1.

Each POW is therefore turned into a total of 23 numbers describing its shape, size and position.

Also, the following whole word features are stored:

- Image proportions.
- The number of dots in the image.
- A resized version of the word image (16×64 pixels).

This is repeated for each image in the training set, and the POW features are stored along with the class label for each POW.

If the whole word features are being used, these features for the words in the training set are grouped by class. The mean of the resized word images is calculated for each class. Two Gaussians are fitted to the range of image proportions and dot numbers in each class.

To summarise, the features used are:

F_1 : The series of angles between the tracing steps of each POW.

F_2 : The centroid of each POW.

F_3 : The length of the line forming each POW.

W_1 : Proportions of each town name image.

W_2 : The number of dots in each image.

W_3 : A resized version of the word image (16×64 pixels).

5.4.2 Classification

The images in the test set are then traced one by one, and features extracted as in Section 5.4.1.

5.4.2.1 POW Classification

For every image, each POW is compared to every POW in the training set. The features of the POW are individually compared and turned into pseudo-probabilities as follows:

The Euclidean distance between the test feature vectors and all those in the training set are calculated. These distances are then converted into pseudo-probabilities using:

$$P(\text{POW}^i | F_1^i, F_1^{\text{test}}) = \frac{e^{-D_1^i}}{\sum_i e^{-D_1^i}} \quad (5.1)$$

where

$P(\text{POW}^i | F_1^i, F_1^{\text{test}})$ Pseudo-probability that POW^i in the training set is the best match to POW^{test} , based on comparing F_1^i and F_1^{test} .

F_1^i Feature vector of POW^i , obtained using feature F_1 .

F_1^{test} Feature vector of POW^{test} , using feature F_1 .

D_1^i Euclidean distance between F_1^i and F_1^{test} .

This is done for each of the three feature vectors of the test POW. The three sets of pseudo-probabilities are then combined for each POW as follows:

$$P(\text{POW}^i | F_{1-3}) = \frac{1}{Z} P(\text{POW}^i | F_1) P(\text{POW}^i | F_2) P(\text{POW}^i | F_3) \quad (5.2)$$

where

$P(\text{POW}^i | F_{1-3})$ Pseudo probability of POW^i being the best match in the training set, given features $F_1 - F_3$.

$P(\text{POW}^i | F_1)$ Abbreviated version of $P(\text{POW}^i | F_1^i, F_1^{\text{test}})$.

Z normalising factor.

This is a similar idea to the Naive Bayes approach, where a strong independence assumption is made.

The largest M probabilities are selected, and those in the same class are summed. This is analogous to a weighted M -nearest neighbour approach. This produces a

vector where each class has a certain value associated with it.

$$P(C_j|\text{POW}^k) = \frac{1}{Z} \sum_{k=1}^M I[C_{\text{POW}^k} = C_j] P(\text{POW}^k|F_{1-3}) \quad (5.3)$$

where

$P(C_j \text{POW}^k)$	Pseudo-probability of POW^k being in class C_j .
$I[C_{\text{POW}^k} = C_j]$	Indicator function. $I = 1$ if class of POW^k is C_j , and 0 otherwise.
$P(\text{POW}^k F_{1-3})$	Pseudo probability of POW^k being the best match in the training set, given features $F_1 - F_3$.
Z	normalising factor.

5.4.2.2 Whole-word Classification

The above process is performed for each of the N POWs in the test word, and the class value vectors summed and renormalised. Each class now has a certain pseudo-probability allocated to it (Equation 5.4).

$$P(C_j|F_{1-3}) = \frac{1}{Z} \sum_{g=1}^N P(C_j|\text{POW}^g) \quad (5.4)$$

where

$P(C_j F_{1-3})$	Pseudo-probability of test town name being in class C_j using features $F_1 - F_3$.
$P(C_j \text{POW}^g)$	Pseudo-probability of POW^g being in class C_j .
Z	normalising factor.

If whole-word features are to be used, they are added at this point by multiplying the contributions from each whole word feature by our class pseudo-probabilities as follows:

$$P(C_j|\text{TestName}) = \frac{1}{Z} P(C_j|F_{1-3}) P(C_j|W_1) P(C_j|W_2) P(C_j|W_3) \quad (5.5)$$

where

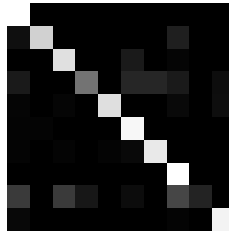


Figure 5.6: *Confusion matrix for the improved tracing method. True labels run down the left hand side, and predicted labels along the top.*

$P(C_j \text{TestName})$	Pseudo-probability of test name being in class C_j .
$P(C_j F_{1-3})$	Pseudo-probability of test name being in class C_j using features $F_1 - F_3$.
$P(C_j W_1)$	Pseudo-probability of test name being in class C_j using whole-word feature W_1 .
Z	normalising factor.

Of the whole-word features, the pseudo-probability $P(C_j|W_1)$ is calculated by comparing the resized test image to the class means of the resized training images (as in Section 3.3). These distances were then turned into pseudo-probabilities in a similar way to Equation 5.1. The other two features, number of dots (W_2) and image proportions (W_3), had Gaussians fitted to the training data. The values of the test image features could then be used to directly generate probabilities. The number of dots can be considered as an Arabic-specific feature, since Arabic has sufficient dots to make this a useful feature. This may not be the case for other writing systems.

The result is a vector with a pseudo-probability allocated to each class. The maximum value is found and the corresponding class allocated to the test word.

5.4.3 Results

Using only the POW features, and using $M = 5$ nearest neighbours, the performance on the small testset was 88%. The confusion matrix for this improved method is shown in Figure 5.6. When the whole-word features were added, per-

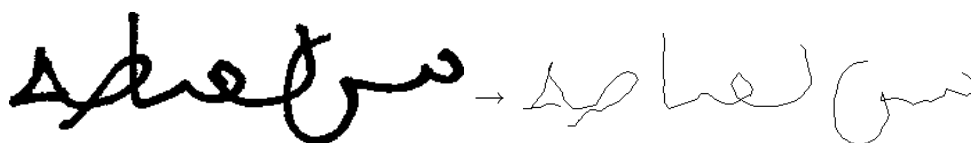


Figure 5.7: *The tracing algorithm resolving overlap between three POWs.*

formance increased to 94%.

This level of accuracy encouraged testing on the full database, where performance was much lower. On the full database, the Tracing Approach achieved 31% accuracy, and adding whole-word features improved accuracy to 50%.

It was noted however, that the algorithm performed better on the classes with more numerous training examples. The average performance of those classes with over 100 training examples was then calculated, and found to be much higher at 74%, or 78% including the whole-word features.

Some tests were also done on using the tracing algorithm to separate overlapping POWs. An example is shown in Figure 5.7. It can be seen that the tracing algorithm effectively separates these three POWs.

5.5 Discussion

The performance reported in Section 5.4.3 above is encouraging, especially since the method has so far been implemented in a rather basic form. There is still much room for improvement in both the tracing algorithm and the formation of feature vectors and classification. There is also the possibility of using more POW and whole word features, and applying more advanced classification algorithms.

The high performance of the Tracing Approach on classes with over 100 training examples is particularly encouraging. Once the algorithm has seen enough training examples, good performance is achieved even for a large lexicon.

This suggests that the Tracing Approach is best suited to applications where each class has a significant number of training examples. It would be interesting to

test performance on a database made up of only the data for classes with over 100 training examples. This would be a 32 class problem, and performance of at least 80% could be expected.

Chapter 6

Discussion

6.1 Introduction

So far in this thesis we have seen a progression from simple models using whole image features to more specialised models.

We have seen how methods using whole-word features can be effective where we have a small lexicon. The performance of the Tangent features method on our small test set, consisting of ten classes, was 94%. This figure would be acceptable for some applications (e.g. cheque processing, which has a small lexicon) and could be improved upon.

Performance of the same methods on the full database shows a more intelligent method is needed for applications with a large lexicon.

The Tracing method is an example of a more intelligent approach, which is shown to have promising performance, along with other advantages such as reconstruction of temporal information and overlap separation.

6.2 Summary of Results

Tables 6.1 and 6.2 give a summary of the best results obtained by different recognition methods during this project. It can be seen that the best performing methods are the Tangent Features and Tracing Approaches. Table 6.3 summarises the best PCA baseline result, and compares it to other methods.

<i>Method</i>	<i>Best accuracy on small testset</i>
Nearest Neighbour – § 3.2	81%
Class Means only – § 3.3	90%
Tangent Features – § 3.3	94%
Radial Method – § 4.3	52%
Hu Moments – § 4.4	35%
Zernike Moments – § 4.4	80%
Tracing Approach – § 5.4	88%
Tracing Approach with whole-word features – § 5.4	94%

Table 6.1: *Summary of results on the small testset using different methods.*

<i>Method</i>	<i>Best accuracy on full database</i>
Class Means only – § 3.3	50%
Tangent Features – § 3.3	52%
Tracing Approach – § 5.4	31%
Tracing Approach with whole-word features – § 5.4	50%

Table 6.2: *Summary of results on the full database using different methods.*

<i>Method</i>	<i>Percentage acceptable (≤ 7 pixels)</i>
PCA – § 2.3	82%
Hough Projection [33]	83%
Skeleton-based [33]	88%

Table 6.3: Summary of best baseline estimation performance.

6.3 Baseline Estimation

One of the main advantages in using PCA over the other methods mentioned in Section 2.2 is that the baseline angle can be found quickly in just one simple step. The other methods require either multiple steps (such as the projection histogram) or more complex calculations (e.g. Hough Transform and skeleton-based methods).

For instance, the PCA method has almost the same performance as the method using Hough Transforms, and yet is a simple two stage process. The Hough Transform on the other hand, requires an examination of all the lines going through all the datapoints at various different angles, and is therefore computationally expensive.

Also, as mentioned in Section 2.5, the PCA method has many improvements that can be made, e.g. examining the two most significant eigenvectors to check which one gives the best estimate.

6.4 Tangent Features

An advantage of the Tangent Features method is that it works by generating more representative training points, and can therefore be used prior to any learning algorithm that accepts training data in vector form.

The method is a way of taking the training set and using it to generate a smaller training set, one that allows better generalisation and faster training time. It

would be interesting to try this method on other similar machine learning problems to see what improvement it can make.

By viewing the eigenvectors it is possible to see what the most significant variations in each class are. This makes is a potentially useful method for data visualisation.

6.5 Moment Invariants

Section 4.5 considered one of the main sources of error to be the inability of the Zernike Moments method to be invariant to local variations.

Unlike the moment invariant approach, the Tangent Features of Section 3.3 can partially account for these local variations. The variations are found from the training data itself, rather than chosen by the system designer. The eigenvalues approach allows these variations to be effectively described by an image, and can therefore describe variations at the level of a small group of pixels. This explains why the Tangent Features performance is higher than that of the Zernike Moments.

6.6 Tracing Approach

There are notable advantages to the approach of tracing words to be recognised:

1. Only the important information about the pen lines is retained, allowing a more compact description of the town name we wish to recognise.
2. Some temporal information about how each POW is formed is recovered by this tracing process. It would be interesting to investigate what advantages this might have for off-line signature analysis.
3. The tracing approach also has the ability to deal with overlap in POWs (see Figure 5.7), something which can be difficult when analysing a word

in image form.

4. Because the tracing algorithm concentrates on following lines, it can be applied without alteration to handwriting in many different languages.

Despite the simplistic approach taken at many stages of the Tracing method, performance is already sufficiently high to consider using it for small lexicon problems (e.g. cheque processing). Performance on large lexicons needs improvement before it will be acceptable for most applications. However, there is still plenty of potential for improvement, particularly in the feature extraction stage. Results on common classes of the IFN/ENIT database suggest that performance improves dramatically when a larger number of training examples are available for each class.

6.7 Contribution to the field

I believe the main contribution of this work to the field of Arabic Handwriting Recognition has been in three major areas:

1. The application of Principal Components Analysis (PCA) to finding the baseline of Arabic town names. Performance of PCA was encouraging, being similar to that of other methods. However, PCA has the advantage that the baseline angle is found in just one step, unlike most of the methods discussed in Section 2.2.
2. The formulation of the *Tangent Method*, where common variations in the data are identified using PCA, and used to generate a smaller, more representative training set. This method allows improved generalisation, whilst reducing training time and storage requirements. It can also be applied as a precursor to many similar machine learning tasks.
3. The use of the *Tracing Approach* to form a compact internal representation of off-line handwriting data, and its use for word/name recognition. Its ability to partially recover temporal data about the formation of a word

suggests it might also be useful in applications such as off-line signature analysis. Another desirable property of the method is its ability to deal with overlapping lines and word fragments (see Figure 5.7).

6.8 Conclusions

- A baseline-finding algorithm using Principal Components Analysis was implemented, including methods to deal with the influence of dots, ascenders and descenders. This was applied to the IFN/ENIT database, with results comparable to other, more complex methods.
- A variety of approaches for the classification of Arabic Town names in the IFN/ENIT database were investigated, and their performance compared.
- The Tangent Features method was formulated using common transformations inferred from the data, and was found to improve generalisation ability whilst reducing time and storage requirements. It gave high performance on a small-lexicon problem, but struggled to achieve high performance with a large lexicon.
- Two classification methods using invariant features of the images were implemented. The Radial Method used distance of pixels from the image Centroid, but gave poor performance as too much information was lost. A Zernike Moment description of the image performed much better, equalling the performance of the Nearest Neighbour approach, but using less features.
- A Tracing Approach was developed where the town name image was traced to produce a pseudo-on-line representation. The image was also split into connected components to aid classification. This scheme had the desirable properties of reconstructing temporal information, and being able to deal with overlapping segments. Performance was found to be encouraging, but the algorithm would need to be improved before use on large lexicon problems could be considered.

6.9 Future Directions

Since handwriting recognition is such a large subject, there is plenty of scope for future directions. If more time had been available it would have been interesting to look at some of the following ideas:

1. The Tracing Approach described in Chapter 5 shows promise and would benefit from an improved algorithm. There is also plenty of scope for different ways of converting the trace into a feature vector.
2. Given the improved performance of the Tracing approach on classes with over 100 training examples, it would be interesting to test performance on a dataset containing just those classes. From the initial results in Section 5.4.3, good results can be expected.
3. With the exception of the parts of Chapter 5 dealing with Arabic specific features (i.e. the number of dots), all the methods described in this report could be applied to handwriting in other languages. Because the tracing algorithm simply deals with following the pen line, it could be applied to any written language without alteration. Indeed, it would be interesting to use the tracing algorithm in signature analysis, since it can recover some of the information about how the signature was formed.
4. It would also be interesting to compare the performance of different machine learning methods once the feature vectors have been formed. This was not a major focus of this project since how we deal with the representation of information from images is likely to make more of a difference in performance. This has been the case in many other machine learning tasks, e.g. [3].
5. Because the Tangent Features method generates a smaller training set of vectors with better generalisation from a larger set, it can be applied to many similar machine learning tasks. Testing on other datasets for different problems may lead to beneficial results.

Bibliography

- [1] A. Amin. Recognition of printed arabic text based on global features and decision tree learning techniques. *Pattern Recognition*, 33(8):1309–1323, August 2000.
- [2] R. M. Bozinovic and S. N. Srihari. Off-line cursive script word recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(1):68–83, 1989.
- [3] Michael C. Burl, Lars Asker, Padhraic Smyth, Usama M. Fayyad, Pietro Perona, Larry Crumpler, and Jayne Aubele. Learning to recognize volcanoes on venus. *Machine Learning*, 30(2):165–194, 1998.
- [4] Richard G. Casey and Eric Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(7):690–706, 1996.
- [5] M.-Y. Chen, A. Kundu, and S. N. Shihari. Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE Transactions on Image Processing*, 4(12):1675–1688, 1995.
- [6] C.W. Chong, P. Raveendran, and R. Mukundan. Translation invariants of zernike moments. *Pattern Recognition*, 36(8):1765–1773, August 2003.
- [7] M. Dehghan, K. Faez, M. Ahmadi, and M. Shridhar. Handwritten farsi (arabic) word recognition: A holistic approach using discrete HMM. *Pattern Recognition*, 34(5):1057–1065, May 2001.

- [8] Sherif Sami El-Dabi, Refat Ramsis, and Aladin Kuwait. Arabic character recognition system: a statistical approach for recognizing cursive typewritten text. *Pattern Recognition*, 23(5):485–495, 1990.
- [9] A. Elgammal and M.A. Ismail. A graph-based segmentation and feature-extraction framework for arabic text recognition. In *ICDAR'01*, 2001.
- [10] E. J. Erlandson, J. M. Trenkle, and R. C. Vogt. Word-level recognition of multifont Arabic text using a feature vector matching approach. In *Proc. SPIE, Document Recognition III, Luc M. Vincent; Jonathan J. Hull; Eds.*, volume 2660, pages 63–70, March 1996.
- [11] J. Flusser and T. Suk. Pattern recognition by affine moment invariants. *Pattern Recognition*, 26(1):167–174, January 1993.
- [12] J. Flusser and T. Suk. Affine moment invariants: A new tool for character-recognition. *Pattern Recognition Letters*, 15(4):433–436, April 1994.
- [13] A. Gillies, E. Erlandson, J. Trenkle, and S. Schlosser. Arabic text recognition system. In *Proceedings of the Symposium on Document Image Understanding Technology, Annapolis, Maryland, 1999*.
- [14] A. M. Gillies. Cursive word recognition using hidden markov models. In *Proceedings U.S. Postal Service 5th Advanced Technology Conf.*, pages 557–562, 1992.
- [15] M. Gilloux, M. Leroux, and J.-M. Bertille. Strategies for handwritten word recognition using hidden markov models. In *Proceedings Int. Conf. on Document Analysis and Recognition*, pages 299–304. IEEE Computer Society, 1993.
- [16] N. D. Gorsky. Experiments with handwriting recognition using holographic representation of line images. *Pattern Recognition Letters*, 15(9):853–859, 1994.
- [17] Venu Govindaraju and Ram K. Krishnamurthy. Holistic handwritten word recognition using temporal features derived from off-line images. *Pattern*

- Recognition Letters*, 17(5):537–540, 1996.
- [18] Steve Grand. *Growing up with Lucy - How to build an android in twenty easy steps*. Weidenfeld & Nicolson, London, first edition, 2003.
- [19] Didier Guillevic and Ching Y. Suen. HMM word recognition engine. In *Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 544–547. IEEE Computer Society, 1997.
- [20] L. Harmon. automatic recognition of print and script. *Proc. IEE*, 60:1165–1176, 1972.
- [21] A. Hashizume, P.S. Yeh, and A. Rosenfeld. A method of detecting the orientation of aligned components. *Pattern Recognition Letters*, 4:125–132, 1986.
- [22] Y. He, M. Y. Chen, and A. Kundu. Off-line handwritten word recognition using HMM with adaptive length viterbi algorithm. In *Proceedings Int. Conf. on Pattern Recognition*, pages 460–462, 1994.
- [23] M.K. Hu. Visual pattern recognition by moment invariants. *IRE Tras. Inform. Theory IT*, 8(2):179–187, February 1962.
- [24] J. Kanai and A.D. Bagdanov. Projection profile based skew estimation algorithm for jbig compressed images. *IJDAR*, 1(1):43–51, 1998.
- [25] A. Khotanzad and Y. H. Hong. Invariant image recognition by zernike moments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(5):489–497, 1990.
- [26] Alexander G. Mamislatov. On the fundamental theorem of moment invariants. *Bull. Acad. Science Georgian SSR*, 59(2):297–300, 1970.
- [27] Alexander G. Mamislatov. On the construction of affine invariants of n-dimensional patterns. *Bull. Acad. Science Georgian SSR*, 76(1):61–64, 1974.
- [28] Magdi Mohamed and Paul Gader. Handwritten word recognition using segmentation-free hidden markov modeling and segmentation-based dynamic

- programming techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(05):548–554, 1996.
- [29] Hirobumi Nishida. An approach to integration of off-line and on-line recognition of handwriting. *Pattern Recognition Letters*, 16(11):1213–1219, 1995.
- [30] C. Parisse. Global word shape processing in off-line recognition of handwriting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(4):460–464, 1996.
- [31] Theo Pavlidis. A vectorizer and feature extractor for document recognition. *Comput. Vision Graph. Image Process.*, 35(1):111–127, 1986.
- [32] M. Pechwitz and V. Maergner. Baseline estimation for arabic handwritten words. In *Frontiers in Handwriting Recognition*, pages 479–484, 2002.
- [33] M. Pechwitz and V. Maergner. HMM-based approach for handwritten arabic word recognition using the IFN/ENIT - database. In *ICDAR*, pages 890–894. IEEE Computer Society, 2003.
- [34] M. Pechwitz, S. Snoussi Maddouri, V. Maergner, N. Ellouze, and H. Amiri. IFN/ENIT - database of handwritten arabic words. In *Proceedings CIFED'02*, pages 129–136, 2002.
- [35] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):63–84, 2000.
- [36] B. Plessis, A. Sicsu, L. Heutte, E. Menu, E. Lecolinet, O. Debon, and J.-V. Moreau. A multi-classifier combination strategy for the recognition of handwritten cursive words. In *Proceedings Int. Conf. on Document Analysis and Recognition*, pages 642–645. IEEE Computer Society, 1993.
- [37] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pages 267–296. Kaufmann, San Mateo, CA, 1990.
- [38] P. Y. Simard, Y. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propaga-

- tion. *International Journal of Imaging Systems & Technology*, 11(3):181–194, 2000.
- [39] J. C. Simon. Off-line cursive word recognition. *Proc. IEEE*, 80(7):1150–1160, 1992.
- [40] Tal Steinherz, Nathan Intrator, and Ehud Rivlin. Skew detection via principal components analysis. In *Proceedings of the 5th International Conference on Document Analysis and Recognition*, pages 153–156. IEEE Computer Society, 1999.
- [41] Tal Steinherz, Ehud Rivlin, and Nathan Intrator. Offline cursive script word recognition - a survey. *IJDAR*, 2(2):90–110, 1999.
- [42] M. Teague. Image analysis via the general theory of moments. *Journal of the Optical Society of America*, 70(8):920–930, 1980.
- [43] B. Yu and A.K. Jain. A robust and fast skew detection algorithm for generic documents. *Pattern Recognition*, 29(10):1599–1629, October 1996.
- [44] F. Zernike. Beugungstheorie des schneidenverfahrens und seiner verbesserten form, der phasenkontrastmethode (diffraction theory of the cut procedure and its improved form, the phase contrast method). *Physica*, 1:689–704, 1934.