MSU Tech Report MSUCPS:TR96-31

A Comparison of Hidden Markov Model Features for the Recognition of Cursive Handwriting

By

Scott Connell

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Computer Science Department

1996

ABSTRACT

A Comparison of Hidden Markov Model Features for the Recognition of Cursive Handwriting

By

Scott Connell

Due to the difficulty of character segmentation in cursive handwriting recognition, much recent research has turned to segmentation free approaches of word recognition. While techniques of feature extraction for presegmented characters have been thoroughly explored in the literature, an evaluation of features for use with segmentation during recognition techniques remains sparse. The main purpose of this thesis is to provide a comparison of a number of feature extraction techniques applied to the domain of legal amount recognition in bank checks. An experimental system using Hidden Markov Models and a horizontally sliding window is described. Results are presented for the recognition of the entire legal field using a variety of features. Of the experiments presented here, the best results were obtained by concatenating the feature vectors from the present, previous, and next windows and using principal component analysis to reduce the dimensionality of this resulting vector.

ACKNOWLEDGMENTS

I would like to thank the many people at IBM Almaden Research Center for funding, educating, and inspiring me during the course of this research. In particular, I am greatly indebted to Robin Williams, K. M. Mohiuddin, Jianchang Mao, Peter DeSouza, Sandeep Gopisetty, Raymond Lorie, V. P. Riyaz, Tom Trong, and especially András Kornai, who has been a great mentor, resource, and friend to me. The guidance he has provided has been invaluable.

Special thanks go to my advisor, Anil K. Jain, who has provided advice, encouragement, and motivation during the writing of this thesis. I would also like to thank the third member of my thesis committee, Eric Torng.

I have always had a strong bond with my family, and I believe that this has helped me through many difficult tasks in my life, of which this thesis is no exception. To my Mom, Dad, and Sister, I'd like to give special thanks for always being there to provide support.

TABLE OF CONTENTS

LIST OF TABLES **vii**

LIST OF FIGURES **viii**

0 Introduction **1**

1 Document Image Analysis and OCR **5**

1.1 Data Capture 6
1.2 Preprocessing 6
1.2.1 Thresholding 8
1.2.2 Noise Filtering 8
1.2.3 Smoothing 9
1.2.4 Skew Detection 10
1.3 Image Segmentation 11
1.3.1 Region Segmentation 11
1.3.2 Word and Character-Level Segmentation 13
1.4 Normalization 15
1.4.1 Horizontal and Vertical Normalization 15
1.4.2 Slant Normalization 16
1.4.3 Thinning 17
1.4.4 Moment Normalization 17
1.5 Feature Extraction 18
1.6 Recognition 19
1.6.1 Classification by Minimum Distance 19
1.6.2 Classification by Maximum Likelihood 20
1.6.3 Direct Template Matching 21
1.6.4 String Matching 21
1.6.5 Elastic Template Matching 22
1.7 Hidden Markov Models 23
1.8 Postprocessing 23
1.9 Optical Character Recognition 24
1.9.1 Cursive Handwriting 24
1.9.2 Legal Amount Recognition 25
1.10 Summary 26

2 Features **28**
2.1 Template Features . . . 28
2.1.1 Peephole Methods . . . 29
2.1.2 Projections . . . 31
2.2 Structural Decomposition . . . 31
2.2.1 Chain Codes . . . 32
2.2.2 T-Joints, X-Joints, and End Points . . . 32
2.2.3 Height Contour . . . 32
2.3 Series Expansion . . . 34
2.3.1 Moments . . . 35
2.3.2 The Fourier Series . . . 37
2.3.3 The Gabor Transform . . . 41
2.3.4 Wavelets . . . 41
2.3.5 Karhunen-Loeve Expansion . . . 43
2.4 Delta Features . . . 45
2.5 Summary . . . 45

3 Hidden Markov Models **47**
3.1 Introduction . . . 48
3.2 Model Representation . . . 51
3.3 Model Evaluation . . . 53
3.4 Estimating Model Parameters . . . 55
3.5 Picking an Optimal State Sequence . . . 58
3.6 Observation Densities . . . 60
3.6.1 Discrete Densities and Vector Quantization . . . 61
3.6.2 Continuous Densities . . . 63
3.6.3 Semi-Continuous Densities . . . 65
3.7 Lack of Training Data . . . 66
3.8 Initial Parameter Values . . . 67
3.8.1 Model Topologies . . . 69
3.8.2 Tied Parameters . . . 71
3.8.3 Multiple Feature Sets . . . 71
3.9 2D HMMs . . . 73
3.10 Language Models . . . 74
3.11 Summary . . . 75

4 Character Recognition **76**
4.1 Introduction . . . 76
4.2 Preprocessing . . . 77
4.3 Features . . . 78
4.3.1 50 Dimensional Features . . . 78
4.3.2 Contour Direction Features . . . 79
4.3.3 CO/12 and CO/24 . . . 79
4.3.4 LO10 . . . 80
4.3.5 Principal Component Analysis . . . 80

4.4 LVQ . . . 80
4.5 Multilayer Perceptrons . . . 81
4.6 Hidden Markov Model . . . 83
4.6.1 Multiple Mixtures vs. Full Covariances . . . 83
4.6.2 The Effects of Multiple States . . . 84
4.6.3 Tied Mixtures . . . 87
4.7 Check Data . . . 89
4.7.1 Data Collection . . . 89
4.7.2 Image Thresholding . . . 91
4.7.3 Finding the Legal Amount Zone . . . 92
4.7.4 Slant Detection . . . 94
4.7.5 Characteristics of Data . . . 94
4.7.6 Truthing the Images . . . 96
4.7.7 Bootstrapping . . . 96
4.7.8 Two-Pass Slant Detection . . . 101
4.8 Summary . . . 104

5 Word-Level Recognition 106
5.1 The Recognizer . . . 107
5.2 Grammars . . . 107
5.2.1 N7 Grammar . . . 108
5.2.2 N10 Grammar . . . 108
5.2.3 N28 Grammar . . . 111
5.3 Testing . . . 113
5.3.1 Performance Criteria . . . 114
5.3.2 Test Data . . . 116
5.4 Optimizing HMM Parameters . . . 117
5.5 Segmentation Experiments . . . 118
5.5.1 Window size . . . 119
5.5.2 Lowpass Filtering and Derivatives . . . 120
5.5.3 Triplication . . . 122
5.5.4 N-tuplication . . . 123
5.5.5 Edge Features . . . 124
5.5.6 Fourier Features . . . 127
5.5.7 Contour Direction Features . . . 131
5.5.8 Multichannel Models . . . 131
5.5.9 Effects of the Language Model . . . 137
5.6 Classification Experiments . . . 137
5.7 Summary . . . 141

6 Conclusions 144

BIBLIOGRAPHY 147

LIST OF TABLES

4.1 A Comparison of Baseline Techniques vs. Single State HMM 82
4.2 The Impact of More Components in the Mixture 84
4.3 The Impact of PCA 85
4.4 Recognition Rates for Multi-State HMM 86
4.5 The Impact of k vs. m for Comparable Number of Parameters 87
4.6 State Level Supervised Clustering 88
4.7 Comparing Two Different Thresholding Methods 92
4.8 The Difference Between Two Humans on Word Segmentation 95
4.9 Original Word Categories from Truths 97
4.9 (cont'd). 98

5.1 Combined Word Categories for the N7 Grammar 109
5.2 N7 Grammar 109
5.3 N10 Grammar 111
5.4 30 Category Word Models 112
5.5 N28 Grammar 113
5.6 30 Category Classification Results 139
5.7 Increasing the Number of Mixtures 139
5.8 TriCO/12 vs. TriCO/6 140
5.9 Slant Correction Results 141

LIST OF FIGURES

0.1 Writing in Which Character Segmentation is Difficult 2

1.1 Major Steps in Document Image Analysis 7
1.2 The Different Stages of Handwriting 15
1.3 The Legal Amount and Courtesy Amount Zones 26

2.1 Contour Direction Features 29
2.2 Bending Point Features 31
2.3 Ascender and Descender Zones 33
2.4 Two-Dimensional Haar Wavelet Masks 43

3.1 A Simple Left-Right Hidden Markov Model 50

4.1 Supervised vs. Unsupervised Clustering 88
4.2 An Example Check from the Database (confidential data removed) . . 90
4.3 Content Zone Segmentation: Left Side 99
4.4 Content Zone Segmentation: Right Side 100
4.5 Content Zone Segmentation: Both Sides 100
4.6 Distribution of Slants Measured 102
4.7 Slant Differences from Human 103

5.1 30 Category Word Counts for Different Training Set Sizes 112
5.2 Testing on Two Different Sets 114
5.3 The Effects of Using Different Sliding Windows 119
5.4 Derivatives and Lowpass Filtering 121
5.5 Triplication 123
5.6 Triplication 124
5.7 The Effects of Quintuplication and Septuplication 125
5.8 Edge Features 126
5.9 FFT Feature Calculation 128
5.10 FFT of Features 128
5.11 FFT Triplicated 129
5.12 FFT of Triplicated Features 130
5.13 Triplicated FFT of Triplicated Features 130
5.14 Contour Direction Features 132
5.15 Edge Features in 2nd Channel 133
5.16 Triplicated Edge Features in 2nd Channel 134

5.17 Comparison of Two Channel HMMs using Edge Features 134
5.18 Combining CO/12 and FFT Features in 2 Channels 135
5.19 Combining CDF88 and CO/12 Features in 2 Channels 136
5.20 Combining CDF88 and FFT Features in 2 Channels 136
5.21 Using Bigram Probabilities 138

CHAPTER 0

Introduction

The field of *optical character recognition* (OCR) is one of the oldest in the history of pattern recognition by computers (Glauberman [1956]; ERA [1957]). In spite of its age, the state of the art in the field has only reached the point of practical use in recent years. While production systems now exist that are capable of recognizing characters that can be reliably segmented, other more difficult to segment forms of writing remain an open problem (see Figure 0.1). The most troublesome of this type of writing style is cursive handwriting. Following the trends in the development of speech recognition, much of recent research has turned away from attempting to presegment cursive characters, and instead has concentrated on segmentation free approaches (Simon [1992]; Kaltenmeier et al. [1993]). Hidden Markov Models are very well suited for this task, and have become the dominant approach to speech recognition (Baker [1990]; Bahl and Jelinek [1975]; Bahl, Jelinek and Mercer [1983]). Subsequently, this approach has become an important area of research for cursive handwriting recognition, particularly in the dynamic (pen-based) domain (Farag [1979]; Nag, Wong and Fallside [1986]; Nathan et al. [1993]).

While many good feature sets have been developed for the recognition of discrete, segmented characters, it is not at all apparent these features will work as well when consistent segmentation is not available. New feature sets that take advantage of

Figure 0.1. Writing in Which Character Segmentation is Difficult

the dynamic data representation used in Hidden Markov Models may provide better results. However, the search for good features is still an ongoing process. The primary goal of this thesis is to provide a comparison of different feature extraction techniques for the purpose of cursive handwriting recognition using Hidden Markov Models. The main contributions of this work can be summarized as follows:

1. A legal amount recognition system was created using a large database (approximately 20,000 checks) of realistic data. This is the first such system to our knowledge, and therefore the experiments presented here should be more indicative of the difficulties that can be expected in a working production system than experiments performed on more artificial data sets. Section 4.7 will describe the database used for this system.

2. A direct and fair comparison of many different feature extraction methods is presented. This is the first major side by side comparison of different Hidden Markov Model features for handwriting recognition to our knowledge, and as such assists in answering the question of what kind of features are appropriate for this domain. These experiments appear in Sections 5.5 and 5.6.

3. A number of properties are established about Hidden Markov Models in this

domain, including the effects of different methods of parameter allocation, the relationship between the number of trainable parameters and the size of the training set, and the effects of varying the size of the sliding window. These findings can be found in Sections 4.6, 5.5.1, and 5.6.

4. A performance measure, based on the segmentation ability of the model tested, is presented. This measure acts as a good indicator of the model's classification accuracy, while having the advantage of requiring much less data to train. Section 5.3.1 describes this performance measure.

Chapter 1 presents an overview of the field of document analysis with an emphasis on the areas that are relevant to optical character recognition. This will provide the reader with a context into which the field of cursive handwriting recognition fits, and a better understanding of the motivation for this field. Chapter 2 will then discuss techniques of feature extraction that have been applied to the recognition of handwritten characters or words. Special attention will be placed on features that are suited for dynamic representation. Hidden Markov Models and the issues involved in their implementation will be the topic of Chapter 3. Chapter 4 will describe some preliminary experiments that were conducted using discretely written, presegmented characters. This task can be seen as the ideal form of handwriting recognition - that for which the difficult process of segmentation has been eliminated. In contrast, a data set made up of the *legal amount* from bank checks will be presented in the last section of this chapter. Region and word-level segmentation is attempted, making apparent why achieving the performance for the ideal case is very unlikely for this type of real world data. Chapter 5 presents experiments in which a number of different feature sets are compared for the task of word recognition. The legal amount data set introduced in Chapter 4 will be used for this purpose, and results will be first presented based on the system's ability to distinguish important words from unimportant words, and

then later on the task of legal amount recognition. The major conclusions of this thesis will be presented in Chapter 6.

CHAPTER 1

Document Image Analysis and OCR

Over the past few decades, the use of computers in the creation of documents has been of great benefit to society. Software tools such as word processors, computer aided design (CAD) packages, drawing programs, and mark-up languages assist us in the creation of these documents and allow for their storage in a format understood by a computer. In this format a document can be easily edited and high quality hard copies can be created using a printer, or it may be quickly distributed electronically to others across world-wide networks. Additionally, we may want to take advantage of other facilities available to us when the document is in a computer readable or understandable format. Some of these include keyword or pattern searching of what may be very lengthy documents, applying optimization algorithms or simulations on things such as electronic circuit designs, or improving the visual quality of the pages of a book or a photograph by removing noise that could be the result of years of decay.

This is not possible, however, when the present form of the document is on paper. Although we are now in the age of desktop publishing and most recently printed journals and books are originally produced in a computerized format, this is still

not the case for the trillions of old documents, nor the handwritten notes, forms, or drawings, that are still in use by all of us even today. The information contained within these documents must first be extracted from the hard copy and stored in a computerized format if we wish to have the benefits described above. This is the motivation of document analysis. Optical Character Recognition (OCR) is that subfield of Document Analysis which is mainly concerned with the recognition of machine printed or handwritten words or characters in a document. For a more thorough overview of the field of document analysis, see O'Gorman and Kasturi [1995].

Document analysis can be broken into a number of steps as shown in Figure 1.1. Each of these steps will be discussed in turn with a bias toward techniques that are useful to OCR.

1.1 Data Capture

A page of a document is first captured as an image by use of a scanning device and stored as a collection of pixels, often with as many as 300 pixels per inch of the original image. The resulting image may be in black and white in which case the pixels will be stored as a single bit, 256 level greyscale in which they are stored as one byte, or color in which pixels may be stored using three components, red, green, and blue, with each of these components possibly stored as a single byte for a total distinction of over 16 million colors.

1.2 Preprocessing

Preprocessing of an image prepares the image for subsequent recognition by removing some of the irrelevant elements of the image while retaining the relevant information. Common forms of preprocessing include thresholding, noise filtering, smoothing, and

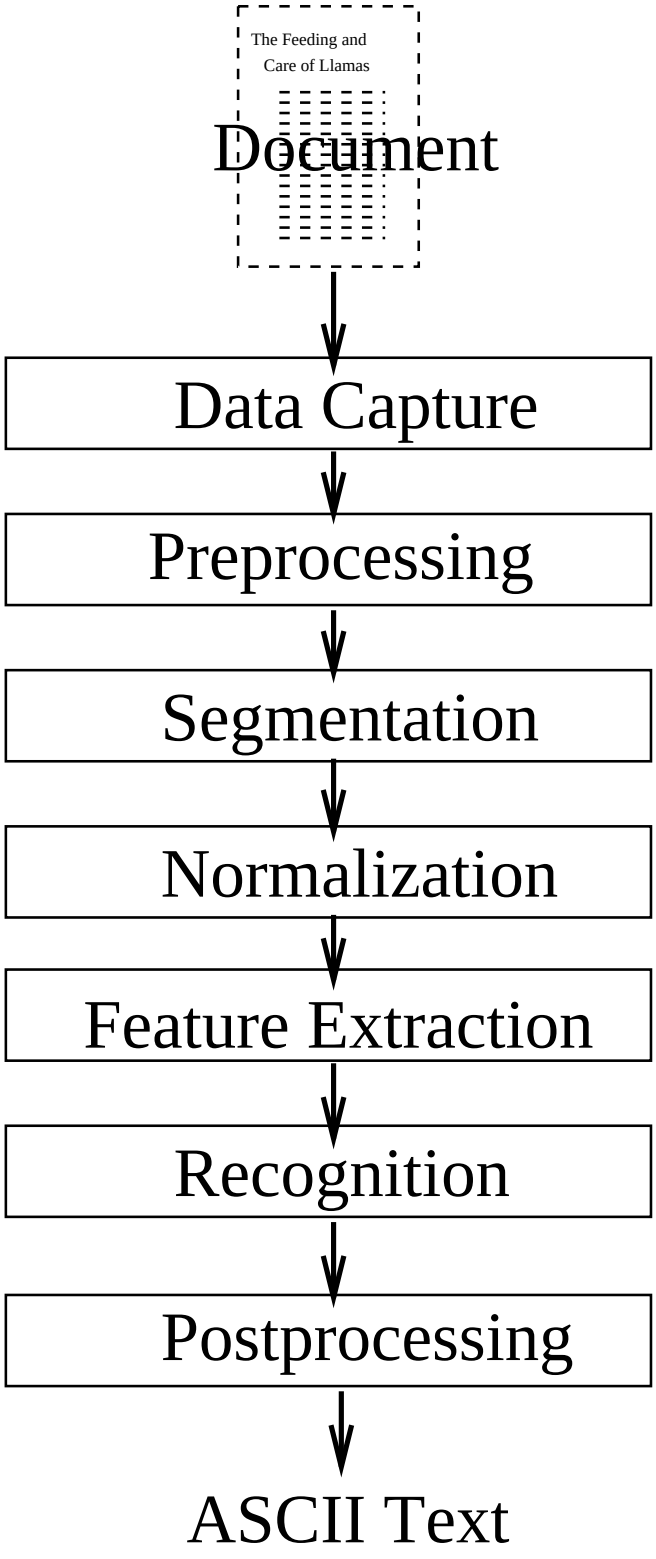

Figure 1.1. Major Steps in Document Image Analysis

skew detection. Each one of these methods will be briefly discussed.

1.2.1 Thresholding

For many applications of document analysis, the full range of greyscale or color images is not necessary. In order to reduce storage requirements and increase processing speed, it is often desirable to represent these images as binary images by picking some threshold value for which everything above that value is set to 1 and everything below is set to 0. The overall goal is to separate objects of interest in the image from the background. Two categories of thresholding exist: *Global* and *Adaptive*. Global thresholding picks one threshold value for the entire image, often based on an estimation of the background level from the intensity histogram of the image. Adaptive thresholding is a method used for images in which different regions of the image may require different threshold values. One way of accomplishing this is by computing a local threshold for each pixel based on a window of its surrounding pixels (Casey and Wong [1990]). Trier and Jain [1995] provide a comparison of many common thresholding techniques using an evaluation criteria that is goal-directed in the sense that the accuracies of a character recognition system using different thresholding techniques were compared. Of those tested, Niblack's method (Niblack [1986]) produced the best results. The check images used in this research were thresholded using an adaptive method (see White and Rohrer [1983]). Section 4.7.2 will describe a method of global thresholding that was developed, and will compare its results to that of Niblack's method.

1.2.2 Noise Filtering

The process of scanning is not exact. Even bi-level documents that are scanned, when digitized as greyscale or color, show more than two levels of darkness. This may be

due to variances in the printing or reflectance of the document, or imprecisions in the scanner itself. Because of factors such as these, it is difficult to pick a good threshold. The effect of setting the threshold too low is the presence of spurious black pixels throughout the image, often known as *speckle noise*. This noise should be eliminated from the image as much as possible so as not to cause confusion during recognition. One technique for eliminating speckle noise is to use a *sigma filter* (Lee [1983]) which averages each pixel with the pixels in its neighborhood. Other methods of noise removal can be found in Haralick and Shapiro [1992]. In our experiments no explicit noise filtering was done. However, in the preparation of one data set, the binarization threshold was set to an empirically determined global value such that speckle noise was kept to a minimum (see Section 4.2).

1.2.3 Smoothing

The opposite problem of that described above is setting the threshold too high. This results in an image in which lines may be faded or broken. Broken lines can be reconnected and jagged lines can be smoothed by a set of morphological operations known as *opening* and *closing* (Haralick and Shapiro [1992]). Chen, Kundu and Zhou [1994] accomplished connecting narrow channels and thin lakes (holes) in cursive writing by the use of *closing*, while *opening* was used to smooth contours and suppress small isolated islands of blackness. The size of these morphological operators must be adjusted so as not to eliminate desirable small effects such as dots on characters like i or j, or holes in letters such as e or g. Smoothing was not a technique used in our research.

1.2.4 Skew Detection

An additional form of preprocessing required for text documents is *skew detection*. Due to inaccuracies in the scanning process, the document may be slightly tilted within the image. This can hurt the effectiveness of later algorithms and therefore should be corrected. Methods of detecting the amount of skew include using the *projection profile* of the image, using the *Hough transform*, and a form of *nearest-neighbor* clustering.

The projection profile method creates a profile histogram by projecting lines at angles around and including the horizontal direction. The angle for which the histogram shows the largest maximum variation between peaks and valleys is taken as the correct angle (Postl [1986]).

The Hough transform is a method of line detection that maps (x, y) space to an accumulator space (r, θ), where θ is the angle of a line found in the image and r is the distance of the line from the origin (Duda and Hart [1972]). This can then be used to find the dominant angle for which the accumulator has high counts, and we can set this as the skew angle. Proper tuning of the resolution of r must be done so as to compute the skew from objects of the desired size (e.g., from columns of characters), thus forcing the size of these objects to be known ahead of time. Techniques using run-length encoding have been proposed to overcome this problem (Hinds, Fisher and D'Amato [1990]). Yu and Jain [1996a] have applied the Hough transform to the centroids of connected components to obtain a robust skew detection technique.

Nearest-neighbor methods involve segmenting the image into components and computing the nearest neighbors for the center of gravity of each of these components. This has the effect of connecting characters within words since they are closer together than cross-word character distances. An angle histogram of the direction vectors connecting these characters can be created and the peak of the histogram taken as

the skew angle (Hashizume, Yeh and Rosenfeld [1986]). In our work we used a variant of the Postl method (see Section 4.7.3).

1.3 Image Segmentation

By segmenting the image into separate components, such as graphics, pictures, and text, each segment can be further processed using different techniques tuned to the type of data contained in that component. In this way not only can a document analysis system focus on recognition for the type of data contained within a component, but different compression algorithms can be used on these components to minimize the overall storage requirements for the document. At a finer level, the field of *optical character recognition* further addresses the problem of word-level segmentation and character-level segmentation. This section discusses some of the issues of segmentation, first at the higher level, which will be referred to as *region segmentation*, and then at the finer levels of which *word segmentation* and *character segmentation* are a part.

1.3.1 Region Segmentation

Two extremes of region segmentation exist that are defined based on the amount of prior knowledge that exists about the structure of the document. These can be categorized as *page layout analysis* and *forms processing*. Page layout analysis is concerned with the segmentation of different regions of a page of text (such as paragraphs, titles, rows, etc.) with little or no prior knowledge of the layout and contents of the document and relies on top-down and bottom-up approaches to reconstruct the structure of the image before individually processing these components. The field of forms processing works with documents for which layout information is available ahead of time, and the job of defining individual components of the image is reduced

to finding some registration points on which the structural definition can be aligned.

Page Layout Analysis

Two stages to page layout analysis have been defined (O'Gorman and Kasturi [1995]). The first, *structural analysis* is concerned with the segmentation of the image into blocks of document components (paragraphs, rows, words, characters, etc.). The second, *functional analysis*, uses location, size, and other layout rules to label the functional content of document components (title, abstract, etc). Structural analysis may be performed in one of the following ways: a *top-down* approach in which the image is broken into smaller and smaller subgroups, a *bottom-up* approach in which first the smallest components are segmented and then grouped into larger and larger components, or some combination of the two approaches. The top-down approach is often accomplished by a run-length smoothing algorithm which smears together components that are within a predefined distance of each other (Wong, Casey and Wahl [1982]). This distance is set depending on the size of the components contained in the regional blocks we wish to segment. Projection profiles can then be used to find and segment the blocks containing these components. Many bottom-up approaches also use this run-length smoothing-projection profile technique to divide the image into regions in which connected-component analysis can be performed. Statistics on these connected-components are then used to group them into larger and larger document components (Fisher, Hinds and D'Amato [1990]).

Forms Processing

Forms processing (see Yu and Jain [1996b]) can involve techniques from many of the different areas of document analysis discussed so far. One characteristic of forms processing is that there exists detailed prior knowledge about the structure of the

document such as the size and location of fields. The goal here is to extract specific information from the document that has often been written there by a human. Information known about the form can be used to find the different fields and then perform recognition on the extracted contents. An example of this could be a table in which the number of columns and rows are known, the approximate size of the table is known, and the location of the cells in the table are known in reference to the edges of the table and to neighboring cells. With the use of line detection techniques, such as previously described, and location information, the lines used to separate cells can be removed (a process known as *form dropout*) and the coordinates where the cell contents reside can be defined. The layout of forms tends to be somewhat rigid, often reducing the problem to that of finding some registration points to detect the form location and scale, and performing dropout. Recognition can then be done on each field, taking into account the type of information that is expected to be found in that particular field.

1.3.2 Word and Character-Level Segmentation

Due to the large amount of space separating words in most texts, segmentation of words can usually be performed by looking for valleys in the vertical projection profile. Another method is connected-component analysis, where word boundaries can be defined based on the distance between two connected components.

Casey [1995] provides a good survey of character segmentation techniques. The difficulty in character segmentation ranges from that of machine printed discrete characters to the more difficult problem of recognizing words and symbols that have been handwritten by a human. Tappert [1984] defines five different stages of the handwritten word problem:

1. Boxed discrete characters

2. Spaced discrete characters

3. Run-on discretely written characters

4. Pure cursive script writing

5. Mixed cursive, discrete, and run-on discrete

Examples of each are shown in Figure 1.2 which reproduces Figure 1 of Tappert [1984].

The first item on this list requires the writer to place each character within its own box on a form. The boxes themselves can be easily found and dropped out of the image, or can be printed on the form in a special color ink that will not be picked up during scanning, thus eliminating the segmentation problem entirely. Spaced discrete characters can be segmented reliably by means of horizontal projections, creating a histogram of grey values in the image over all the columns, and picking the valleys of this histogram as the points of segmentation. This has the same level of segmentation difficulty as is usually found with clean machine printed characters. Characters at the third stage and below are usually discretely written, however they may be touching (stage 3), therefore making the points of segmentation less obvious. Degraded machine printed characters may also be found at this level of difficulty. These require more sophisticated methods to find points where characters are connected and break them apart. Even more difficult are the problems of cursive handwriting and mixed cursive and discrete writing, which will be discussed in Section 1.9.1.

The type of segmentation that has been described so far precedes recognition, therefore the recognition process is very dependent on the components handed to it from the segmenter. Incorrect segmentation (i.e., splitting a character or grouping two characters) destroys any chance the recognizer may have of finding the correct identities of the characters. Postprocessing may attempt to remedy this, but the

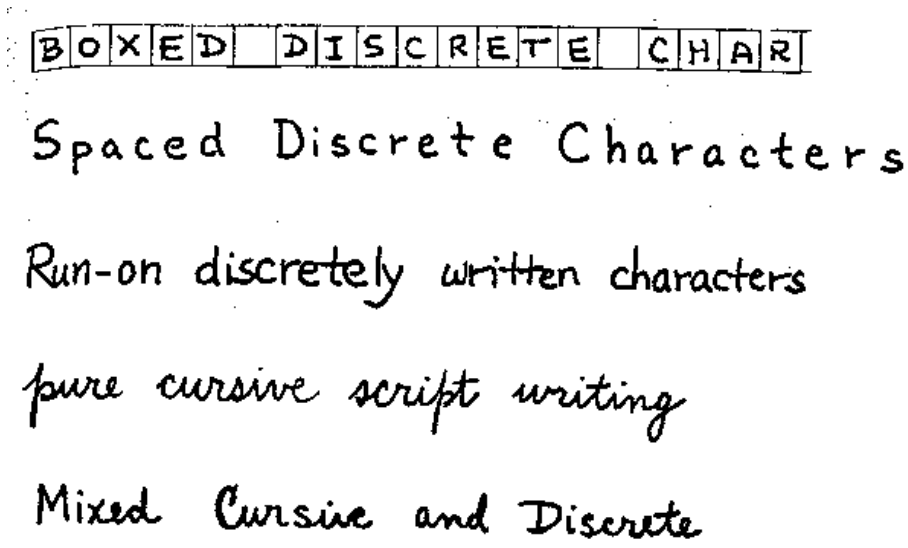

Figure 1.2. The Different Stages of Handwriting

fact remains that the weakness is occurring early on in the system. Another option, concurrent segmentation and recognition, will be further discussed in Section 1.9.1.

1.4 Normalization

The goal of normalization is to remove intra-class variances, while retaining as much inter-class variance from the original image as possible. Types of variation that exist independent of class identity include horizontal and vertical size, the slant of the writing, the thickness of the written strokes, and moments.

1.4.1 Horizontal and Vertical Normalization

The simplest form of normalization is size normalization. If a character is written exactly the same twice with the exception of one character appearing much taller or wider than the other, we want to ensure that both characters are still classified the

same, and with a high degree of confidence. Differences in the size of characters written should be eliminated so that it will not be a factor during classification. Methods of character recognition may apply both horizontal and vertical size normalizations after character segmentation. Yanikoglu and Sandon [1994] use a technique to normalize segmented characters in which the character is divided into a number of zones and each of these zones is separately scaled. On the other hand, word recognition, due to the desire to preserve large intra-class differences in the length of words so they may assist in recognition, tends to only involve vertical height normalization, or bases the horizontal size normalization on the scale factor calculated for the vertical normalization.

1.4.2 Slant Normalization

Another dimension of variance that is very characteristic of human handwriting is the degree of slant applied to the characters formed. In order to allow recognition to be slant independent, the degree of the slant should be detected and corrected. Bozinovic and Srihari [1989] used a slant detection technique which begins by dividing the images into different vertical windows through removal of horizontal lines with long runs. These windows are then further divided horizontally by removing window portions that contain no writing. The slant is estimated based on the center of gravity of the upper and lower half of each window averaged over all the windows. Guillevic [1995] uses an approach in which projection profiles are computed for a number of angles away from the vertical direction. The angle corresponding to the projection with the greatest positive derivative is used to detect the least amount of overlap between vertical strokes, and therefore the dominant slant angle. Our method, a variant of the Hough transform, will be described in detail in Chapter 4.

1.4.3 Thinning

Thinning is a method of reducing an image to a collection of lines whose thicknesses are normalized, thus eliminating stroke width variances between examples. This is very useful when an object (such as a character) is to be represented as some description of a collection of objects such as lines with T-junctions, crossing-junctions, and end points. These and other types of feature extraction methods would be adversely effected by differences in line thickness, therefore it may be desirable to normalize these lines by thinning, often to a single pixel width. In our work we presently use no thinning. A survey of thinning methods is available in the literature (Lam, Lee and Suen [1992]).

1.4.4 Moment Normalization

Moment normalization (Casey [1970]) strives to make the process of recognizing an object in an image size, translation, and rotation independent. *Regular moments* are one of the most common types of moments. A regular moment of order $(p+q)$ can be computed as (Hu [1962]):

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y) dx dy,$$

where $f(x,y)$ is the grey value in the image at location (x,y). Another more common form of this moment is known as the *central moment*, in which x and y are measured in reference to the center of gravity along the x axis and y axis, respectively:

$$\mu_{pq} = \int_{-\infty}^{\infty} (x-\overline{x})^p (y-\overline{y})^q f(x,y) dx dy$$

where:

$$\overline{x} = \frac{m_{10}}{m_{00}}$$

$$\overline{y} = \frac{m_{01}}{m_{00}}$$

One normalization technique (Casey [1970]) performs a normalization using the second order moments, μ_{20}, μ_{02}, and μ_{11}. The first two of these moments correspond to the mean-square deviations about the center of gravity of the image in the orthogonal directions of the x and y axes, while the third moment corresponds to deviations in a non-orthogonal direction. The character can then be size normalized, slant normalized, and rotation normalized by means of a linear transformation on the image which causes both the orthogonal deviations to be equal, and the non-orthogonal deviation to be zero. In other words:

$$\mu_{20} = \mu_{02},$$

$$\mu_{11} = 0.$$

Other uses of moments concentrate on creating feature sets which are moment invariant rather than normalizing for these moment variations. These will be presented in Chapter 2.

1.5 Feature Extraction

Prior to the recognition stage, some set of features are extracted from the image. These features are a reduced representation of the contents of the image which focuses on preserving the characteristics that are most relevant to the task of recognition while eliminating those characteristics that are irrelevant to, or may confuse, the discrimination power of the recognition stage. Extracted features are represented as

a vector of values that are then passed to the recognizer. A collection of different feature sets reported in the literature will be discussed in chapter 2.

1.6 Recognition

Recognition involves classifying each unknown object into one of a finite number of categories. This section will discuss two different approaches to classification: minimum distance classification, and maximum likelihood classification. Three different methods of applying these approaches, template matching, string matching, and elastic template matching, will then be described. For a more thorough discussion of recognition techniques, see Duda and Hart [1973] and Tou and Gonzalez [1974].

1.6.1 Classification by Minimum Distance

If patterns from the same class cluster well in some feature space, using some measure of distance, then we can use these clusters to define a number of linear decision boundaries for the classification of an incoming pattern. One method is to create vector quantized prototypes of the clusters by calculating the cluster centers (see Jain and Dubes [1988]; Anderberg [1973]; Everitt [1974] for clustering techniques). An incoming pattern can then be classified using the cluster whose center is the minimum distance from the pattern over all the clusters. Other methods may choose a pattern that best represents that cluster from the patterns in the cluster, or may pick some subset of those patterns which define the decision boundary. Depending on the distribution of the data within each class, it may be advantageous to create multiple clusters for each class. In such cases where many patterns or cluster centers are stored to represent each class, performing classification based on the K nearest patterns, or on the K nearest centers may provide better results. Common measures of distance are Euclidean distance, Manhattan distance, and Hamming distance. Depending

on the application domain, much more complex distance measures are also used. For example, another distance measure, which is used in Linear Projection Coding (LPC), is called the Itakura-Saito measure (Itakura and Saito [1968]). In LPC, a technique that has been popular in the speech recognition community, a sample of a continuous waveform is encoded as a vector $x = (\alpha, a_1, a_2, \ldots, a_p)$ where the elements a_i are z-transform inverse filter coefficients for an all-pole filter with $a_0 = 1$, and α is the total residual energy of this representation. When this filter is applied to a predefined signal (for example, a white noise signal with constant power spectral density), it will reproduce the original signal. By applying this filter to some other signal, a different residual energy will result. For a discussion of distance measures for speech recognition, see Gray and Markel [1976].

1.6.2 Classification by Maximum Likelihood

Rather than defining a number of linear boundaries, another approach is to estimate the distribution of data, possibly in some parametric form such as a Gaussian distribution, or mixture of Gaussian distributions. *Bayes decision theory* is a method of classifying an incoming pattern based on a comparison of a posteriori probability density functions. Using past examples of features from each class (training data), we can estimate the class conditional densities, $P(x|C)$, where x is a feature vector, and C is the pattern class that produced that feature vector. Using the a priori probabilities of the class, $P(C)$, and the occurrence of the feature vector, $P(x)$ $(= \sum_C P(x|C))$, we can calculate the a posteriori probability for that class using the *Bayes decision rule* (Duda and Hart [1973]):

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)}.$$

The category with the maximum a posteriori probability then defines the classification result. In situations where the a priori class probabilities are equal ($P(C) = \frac{1}{M}$), $\forall C$, where M is the number of classes, the Bayes decision rule simplifies to the *maximum likelihood decision rule*: Assign x to class C is $P(x|C)$ is maximum over all C.

1.6.3 Direct Template Matching

Template Matching is a technique in which a prototype, called a template, is created for every type of pattern that can be presented to the recognizer. Each pattern presented to the system is matched to one of these templates and that template's class defines the classification of the unknown pattern. Template matching can be performed using some minimum distance approach, or a set of likelihood functions can be created for the templates. This method is very inflexible since any form of distortion or noise that may exist in the pattern requires storing a new template. Therefore, in difficult fields such as OCR, this technique is only used in conjunction with other techniques. Some of these techniques will be discussed in chapter 2.

1.6.4 String Matching

Methods of feature extraction such as run length encoding, which uses contour following, produce a string of symbols that may vary in length within the same class due to local and global size variations. The symbols which make up this string may be generated by a form of local template matching (as in run length encoding which will be discussed in chapter 2). The technique of representing a pattern as a string of symbols and comparing this to some class string description is known as *string matching*, or *elastic matching*.

Dynamic programming (Bellman [1957]; Bellman and Dreyfus [1962]) is a method

of finding an optimal stretching or compressing of a string so that it matches as closely as possible some class prototype. Classification can be achieved by minimum distance, or maximum likelihood. Each class contains a prototype string and the incoming pattern string is classified by the prototype which it is matched to using dynamic programming. The minimum distance method associates a penalty with each mismatched symbol in the final string alignment based on some distance measure. It may also assign a penalty based on the amount of stretching or compression that must be imposed on the string. Taking this a step further, rather than using a prototype string of symbols, the likelihood of each symbol can be estimated together with the likelihoods of stretching and compression occurring throughout the string. The product of these likelihoods for the incoming pattern string then produces the likelihood for the entire string.

1.6.5 Elastic Template Matching

Elastic template matching is a method of performing elastic matching to 2-dimensional patterns. The idea is that an instance of a class differs from its prototype template by a number of local distortions. A match can be found between a pattern and its prototype template by applying a number of local transformations to the template. One such transformation method is the local affine transform (Wakahara [1994]). The distance between a template and the pattern being tested can be measured by the amount of distortion between the two. This recognition method is computationally very expensive, and therefore its use has been limited. Jolly, Lakshmanan and Jain [1996] have applied this type of template matching to the task of vehicle segmentation and recognition of traffic images. Applications using more general templates can be found in Jain, Zong and Lakshmanan [1996].

1.7 Hidden Markov Models

In Section 1.6.4 a method of performing string matching based on likelihood functions was described. A Hidden Markov Model is one method of encoding such probabilistic information. Localized parts of the string are represented as hidden states and stretching/compression effects are represented as probabilities of moving from one state to the next, skipping over states, or staying in the present state. The likelihood of observing a symbol from the string in a given hidden state is also estimated. Hidden Markov Models will be discussed in detail in Chapter 3.

1.8 Postprocessing

The goal of postprocessing is to use additional contextual information about the characters or words being recognized to correct possible errors or decide between different candidates passed from the output of the recognizer. This information may take the form of a dictionary lookup that picks from the top scoring word candidates based on which of those found in a predefined dictionary have the highest recognition score. It may be some statistic about common recognition mistakes that may cause letter deletions, insertions, or substitutions within words so that an attempt can be made to fix these problems. A more advanced postprocessing technique is to use a language model, such as a grammar, to define restrictions on the context in which words or characters are used. Such a language model may be probabilistic, involving statistics on the probability of one word following another (bigrams), or n words occurring sequentially (n-grams). Markov Chains are well suited for this type of language modeling since well known dynamic programming techniques can be applied to avoid the combinatorial explosion of tracing all possible string paths. For an overview of the issues involved in Language Modeling, see Kornai [1994].

1.9 Optical Character Recognition

Optical character recognition is divided into two separate domains: recognition of machine printed words or characters, and recognition of handwritten words or characters. Most of the early success in OCR was on machine printed characters and today there are many production systems capable of achieving very good recognition results ($\approx 99\%$) on this type of data. Much success has been achieved in recognizing discrete characters for which segmentation can be done reliably. However, recognition of writing that is difficult to segment, such as cursive handwriting, is very much still an open problem.

1.9.1 Cursive Handwriting

In pure cursive handwriting, a word is formed mostly from a single stroke (line written from pen-down until pen-up). This makes segmentation by the traditional projection or connected-component methods ineffective. Other forms of character segmentation have been attempted which cut the words into a number of preliminary segmentation points which are all passed to the recognizer (Bozinovic and Srihari [1989]; Gader, Mohamed and Chiang [1992]; Yanikoglu and Sandon [1994]) and then the best segmentation is decided by some postprocessing which looks at the recognition confidences over all possible segmentations.

It is with cursive handwriting in particular that the idea of *segmentation during recognition* has become popular. This is often accomplished by use of a sliding window. A window of some predefined size is slid from left to right across the image over (pseudo) time using some predefined step size. At any point in time, recognition is performed on the contents of the window. In this way, a dynamic signal is simulated from a static image. With this in mind, it is useful to look at other fields which recognize dynamic signals. Much of the work presented in the rest of this thesis,

based on Hidden Markov Models, has its roots in the field of Speech Recognition.

1.9.2 Legal Amount Recognition

One application of OCR and document processing that has received much attention lately is the processing of bank checks. A large number of checks (an order of magnitude of 100 million) are processed by banks everyday, and the rate is growing every year. For bookkeeping purposes, the information from these checks must be entered into a computer database. Traditionally, these checks were entered by humans at the cost of millions of man-hours. In order to reduce these costs, banks began to print account numbers on checks in magnetic ink, and to have human operators key check amounts in magnetic ink, so further processing could be done by an automated system. More recently as handwritten digit recognition technology has become very good, systems have been implemented to recognize what is known as the *courtesy amount* on checks. The courtesy amount field is located on the right side of the check, just above the vertical center and is usually marked by a surrounding box or an underline (see Figure 1.3). This field contains digits which designate for what amount the check has been written. Digits are often written discretely spaced and there are a small number of classes to distinguish between, making this field a much easier candidate for recognition than the legal amount field, in which the amount is written in what is often a combination of discrete and cursive writing.

Due to the fact that individually recognized digits do not provide any contextual information, if a digit is recognized incorrectly, there is no means of correcting it in postprocessing. Therefore, since the cost of a recognition error (as opposed to a rejection) is high, the resulting error must be taken seriously. The need for something to cross-validate the courtesy amount has lead to the goal of the more difficult problem of legal amount recognition.

Legal amount recognition is a difficult problem because the writing is a mixture

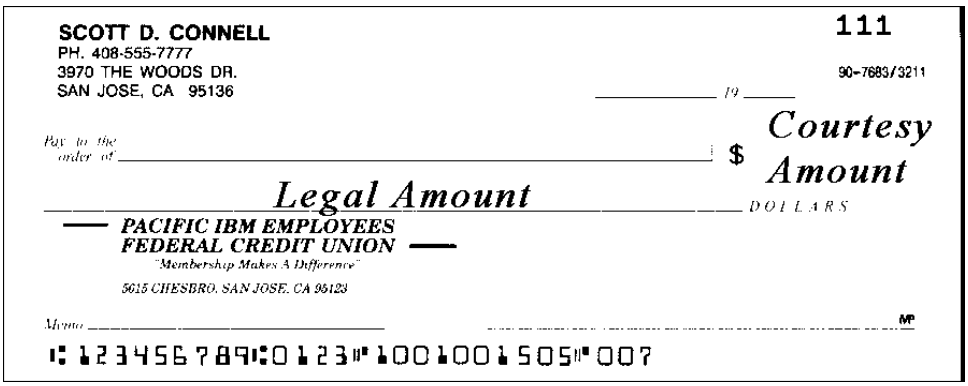

Figure 1.3. The Legal Amount and Courtesy Amount Zones

of cursive and touching or non-touching printed characters. Additionally, other items such as fractions and horizontal lines are usually written, adding to the complexity of objects encountered. One factor that helps make this goal possible is the limited size of the dictionary required - approximately 30 words. However, due to the high amount of between-writer variance, and the incredibly large number of different writers, creating good models for these 30 words can be difficult.

1.10 Summary

We have described the field of document analysis as containing 7 steps: data capture in which the document is scanned, preprocessing in which the image is binarized and noise and document skew are removed, segmentation in which the image is divided into a hierarchy of components that will be worked on individually, feature extraction in which some reduced description of an object's pattern is created, recognition in which that description is used to classify the object, and postprocessing in which higher level restrictions on the classifications of a collection of objects are imposed. Optical character recognition and cursive handwriting recognition are subsets of the field of document analysis that focus on recognizing the text within a document. One

such application, legal amount recognition, has been discussed and will be the main focus of this thesis.

CHAPTER 2

Features

Feature sets play one of the most important roles in a recognition system. A good feature set should represent characteristics of a class that helps distinguish it from other classes, while remaining invariant to characteristic differences within the class. Features should also avoid containing redundant information since this leads to a more complex distribution in the feature space, and therefore requires a more complex model. This chapter will focus mainly on features that are useful for Off-line Cursive Handwriting Recognition as well as the more general field of Optical Character Recognition.

2.1 Template Features

Template matching is a very simple form of pattern matching in which templates are created which are meant to match example patterns pixel by pixel. A template must be formed for each possible input pattern which defines that pattern's classification. Template matching was one of the earliest approaches attempted in OCR research, however its inflexibility has limited its usefulness. In particular, it is very sensitive to size, rotation, and translation variations, as well as to noise. This section describes types of template matching that still have limited use in the field.

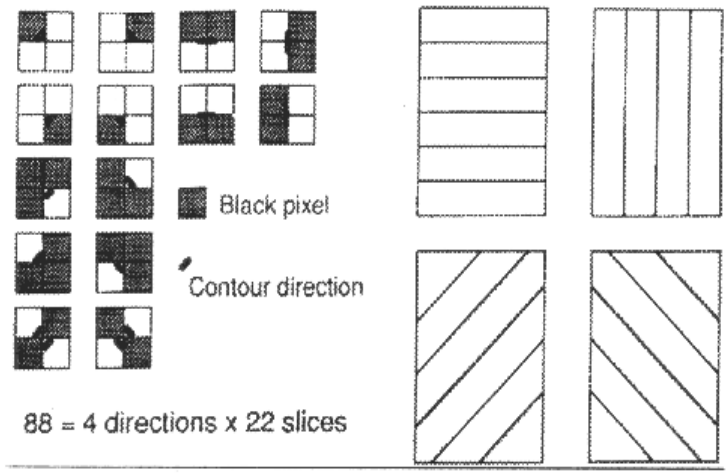

Figure 2.1. Contour Direction Features

2.1.1 Peephole Methods

The *peephole* method is characterized by its measure of localized properties. Rather than using large templates that are matched to the entire image, smaller templates can be used to match to smaller parts of the image. This reduces the number of possible templates that must be stored.

Contour Direction Features

Contour direction features measure the direction of the contours of the characters. This can be done by applying a number of 2×2 masks over the image, one set of masks for each of the four contour directions: vertical, horizontal, and the two diagonals. An example of these masks as used by Takahashi [1991] can be seen in Figure 2.1 which reproduces Figure 2 by Mohiuddin and Mao [1994]. For each 2×2 block of the image, if one of these masks matches this block, then that mask defines the contour direction at the point in between those four pixels.

Takahashi [1991] counts the number of occurrences of each contour direction for use as features. A 24×16 image is divided into a number of zones as shown in Figure 2.1. There are 22 zones total, 4 taken by dividing the image into vertically running strips, 6 from dividing the image into horizontally running strips, and 12 from diving the image into 6 diagonally running strips for both of the diagonals. For each of these zones, the number of occurrences of each contour direction found within that zone are counted, and each count taken as a feature. This creates an 88-dimensional feature vector (22 slices $\times$ 4 directions).

Bending Point Features

Bending points are points at which a stroke in the image has a strong curvature. These can be found by tracing the contour of strokes in the image and matching local parts of the contour against prespecified templates. Takahashi [1991] creates features for single character images from detected bending points by dividing the image into a 4×3 grid of zones as shown in Figure 2.2 which reproduces Figure 3 by Mohiuddin and Mao [1994]. The direction of each bending point is quantized into one of four diagonal directions, and also classified as convex or concave. For each of the 12 zones in an image, the number of each of these 8 types of bending points is summed, for a total of 96 different sums. These are then used in conjunction with the 88 contour direction features described in Section 2.1.1 to create a 184-dimensional feature vector.

Mao, Mohiuddin and Jain [1994] have shown that the entire set of 184 features produces good classification results (96.4%) on the NIST digit database using a neural network. It was also shown that 96 of these features can be eliminated with only a 0.5% reduction in classification accuracy.

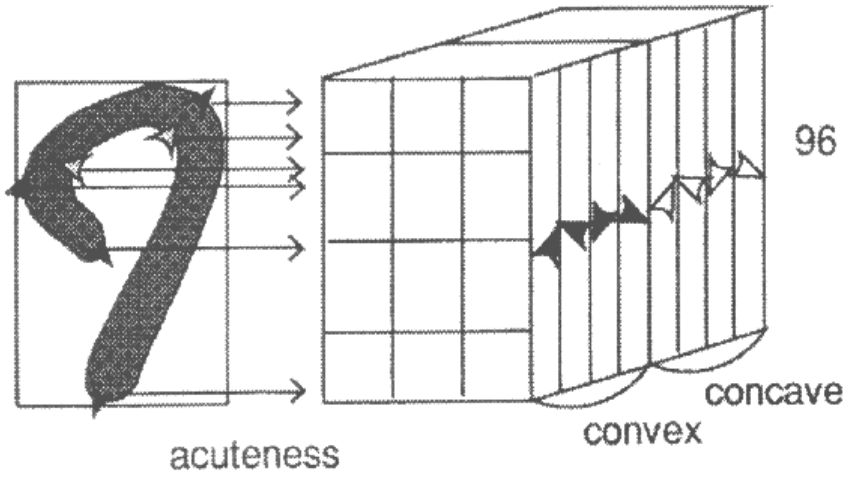

Figure 2.2. Bending Point Features

2.1.2 Projections

Template matching is very dependent on the location of the pattern in the image. This is particularly a problem when the template can be misaligned in the horizontal or vertical directions. One method of reducing this problem is through the use of *projections*. By summing pixels in the horizontal direction or the vertical direction, a one dimensional signal is created which can be used to register the template.

2.2 Structural Decomposition

Structural decomposition is a method of extracting features that tell us something about the structure of the object we are trying to recognize. These types of features may encode some knowledge about the contour of the object, or may require some knowledge as to what sort of components make up that object.

2.2.1 Chain Codes

Chain codes are a method of encoding a curve (see Duda and Hart [1973]). A mesh is placed over the curve and, at each point the curve intersects the mesh, the closest vertices of the mesh are taken to represent a point along the curve. By tracing the curve and listing the direction between consecutive points encountered, the chain code is created. These directions are generally numbered $0-7$ in a counter-clockwise direction with 0 pointing in the eastward direction.

2.2.2 T-Joints, X-Joints, and End Points

These features work on a thinned image. T-joints and X-joints are most likely formed by the intersection of two strokes. A T-joint is distinguished from an X-joint by the termination of one stroke intersecting another stroke somewhere in the middle forming a 'T', while a stroke of an X-joint crosses straight through the intersecting stroke. An end point is a point at which a stroke terminates without touching another stroke, and often may mark the starting or ending of a word or character. Kundu, He and Bahl [1989] find each of these by passing a 3×3 mask over the thinned image. Points where the mask has a black central pixel and 3 neighboring black pixels are labeled as a T-joint, while a central black pixel with 4 or more neighboring black pixels is labeled an X-joint. Subsequently, an end point is found as a central black pixel in the mask with only a single black pixel neighbor.

2.2.3 Height Contour

One method that searches for global features in a word uses the upper and lower height contours of the image to define features. One common approach is to use this contour to label different parts of the word as ascender, descender, or body. In order to find ascenders and descenders, the image is divided vertically into three

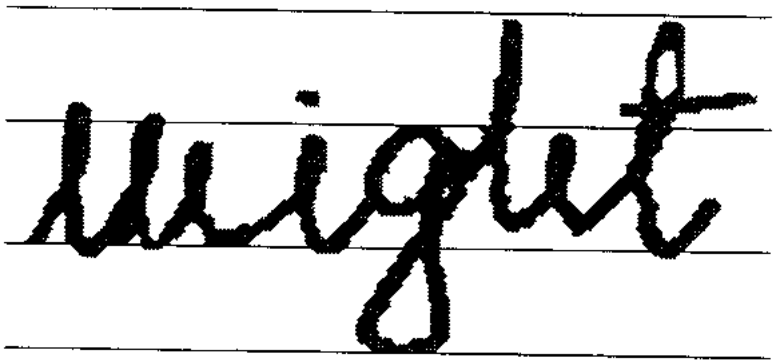

Figure 2.3. Ascender and Descender Zones

zones (see Figure 2.3 which reproduces Figure 2a by Bozinovic and Srihari [1989]). These segmentation points are usually found by creating a horizontal projection profile histogram for the word and finding the central peak which should correspond to the central zone of the word. Upper and lower cut-off points are picked for this peak at the local extrema of the first derivative (Bunke, Roth and Schukat-Talamazzini [1994]), and these then define the zone boundaries. All lines which cross into the upper zone are labeled ascenders and all lines which cross into the lower zone are labeled descenders. Simon [1992] finds ascenders and descenders by smearing the image horizontally. Median filters are then used to find the body of the word, and elements which stick very far outside of the body are taken as ascenders or descenders. Many handwriting recognition systems found in the literature include these types of features. Gilloux, Leroux and Bertille [1993] use the detection of ascenders and descenders, the detection of loops, and the sizes of each to describe the contents of sub-word pieces (approximately one character) segmented from cursive words. These are quantized to 26 symbols plus one symbol to represent whitespace, for a total of 27 symbols. A word is then represented as a train of these symbols running horizontally in the image.

2.3 Series Expansion

A continuous signal generally contains more information than needs to be represented for the purposes of classification. This may be true of discrete approximations of continuous signals as well. One way to represent a signal is by a linear combination of a series of simpler well-defined functions. The coefficients of the linear combination provide a compact encoding known as *series expansion*. If these functions are orthogonal to each other, then the coefficients in their linear combination provide non-redundant information toward the reconstruction of the signal. With no dependencies between these coefficients, the distribution of the data can be represented using much simpler models (e.g., Gaussian diagonal covariances instead of full covariances). In addition, the elimination of any of these functions from the linear combination will only degrade the reconstructed signal in as far as that function contributes to the original signal. Therefore, if certain functions can be identified as unimportant to the task of discrimination, a reduction in dimensions can be achieved by eliminating their corresponding coefficients from the series expanded representation.

One method of feature extraction leading to series expansion, moments, was presented in Chapter 1 and will be further discussed here. Another common technique of series expansion, the Fourier Transform, has been applied to OCR in many ways, while two other methods, the Gabor Transform and Wavelets, have come into popularity only more recently. A final series expansion method, the Karhunen-Loeve expansion, has the property of being optimal in terms of guaranteeing decorrelation of the feature space and therefore minimizing the error introduced when one of the components is eliminated from the signal reconstruction.

2.3.1 Moments

Chapter 1 showed how moments can be used to normalize an object, such as a character, in an image. Alternatively, these moments can be used to form a compact encoding of the original binary image that is invariant to such things as translation, rotation, and scale. Moments are also a form of series expansion in that the original object can be completely reconstructed as a linear combination of all orders of their characteristic polynomial in which the moments are the coefficients in the combination.

Central Moments

In their original form, the central moments (see Section 1.4.4) are sensitive to rotation. Hu [1962] has shown how 7 moment invariant features can be constructed by combinations of the central moments. These are invariant to translation, scale, and rotation, and therefore do not require direct normalization of these properties. Many applications of central moments have been used in tasks where rotation invariance is important, such as aircraft identification (Dudani, Breeding and McGhee [1977]). Kim and Yuan [1994] have applied these moments to the task of rotated character recognition. Central moments are not optimal, however, in that they are not orthogonal and therefore there is a high degree of information redundancy between them. This makes it difficult to reduce the number of moments used to represent an object since eliminating one moment may impact the value of other moments.

Legendre Moments

Regular moments (and therefore Central moments) can be viewed as a projection of the image onto the monomial $x^p y^q$, for $p, q = 0, \ldots, \infty$ (Teh and Chin [1988]). However, the basis set formed by these functions is not orthogonal. Alternatively, other

types of moments can be calculated by projecting onto a different set of functions. The *Legendre moments* are created by projecting onto the orthogonal set of Legendre polynomials.

The n-th order Legendre polynomial is defined as:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n$$

then the $(m+n)$th order Legendre moment is:

$$\lambda_{mn} = \frac{(2m+1)(2n+1)}{4} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P_m(x) P_n(y) f(x,y) dx dy.$$

With such an orthogonal basis as is created by these polynomials, we can reduce the number of coefficients that must be stored by eliminating most of the higher order terms which provide much less information about the structure of the object represented. The orthogonality property means that the elimination of these coefficients will not impact the value of the other coefficients.

Zernike Moments

Zernike moments were proposed by Teague [1980] as a form of orthogonal moments. These are complex-valued moments which are defined using Zernike polynomials:

$$A_{nl} = \frac{n+1}{\pi} \int_0^{2\pi} \int_0^{\infty} [V_{nl}(r,\theta)]^* f(r\cos\theta, r\sin\theta) r dr d\theta$$

where $n = 0, \ldots, \infty$, $|l| \leq n$ and $n - |l|$ is even. The symbol * denotes the complex conjugate.

The Zernike polynomials are a set of functions that are orthogonal on the unit disk $x^2 + y^2 \leq 1$

$$v_{nl}(x,y) = V_{nl}(r\cos\theta, r\sin\theta) = R_{nl}(r)\exp^{jl\theta}$$

where the radial polynomials, $R_{nl}(r)$, are real-valued and defined as:

$$R_{nl}(r) = \sum_{s=0}^{(n-|l|)/2} (-1)^s \frac{(n-s)!}{s!(\frac{n+|l|}{2}-s)!(\frac{n-|l|}{2}-s)!} r^{n-2s}$$

2.3.2 The Fourier Series

One technique of representing a periodic signal that is stationary (its properties do not change over time) is to separately measure the different frequency components of a signal. The Fourier transform takes a signal in the time domain and represents it as its series expansion coefficients in the frequency domain. For a one dimensional time-varying signal, this is accomplished by use of the following equation (Chen [1989]):

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp^{-j\omega t}\, dt$$

for all ω, where ω is a frequency, and $F(\omega)$ is the frequency spectrum over a range of values for ω.

For a digitized signal, the time-domain function is represented as a sequence of discrete time intervals, and discrete amplitudes:

$$f_k = f(k), k = 0, 1, \ldots, N-1$$

over some periodic interval for which N sample points are defined. The Discrete Fourier transform (DFT) is used to convert this into the frequency domain:

$$F(\omega_n) = \sum_{k=0}^{N-1} f_k \exp^{-jnk2\pi/N}$$

where $\omega_n = n\frac{2\pi}{N}$, and ω is taken at discrete intervals specified by $n = 0, \ldots, N-1$. Since the signal $f(k)$ is represented at N discrete intervals per period, no wavelength shorter than one of these intervals (and therefore no frequency greater than ω_N) can be computed from the series.

This transformation can be equivalently described as a projection of the signal onto the orthogonal functions defined by:

$$\phi_n(k) = \exp^{jn\omega_0 k}$$

with base frequency $\omega_0 = \frac{2\pi}{N}$.

A computationally efficient algorithm called the Fast Fourier transform (FFT) exists for calculating Discrete Fourier transforms. Implementations of this algorithm are readily available throughout the literature (see Cormen, Leiserson and Rivest [1990] for one example).

Fourier series expansion has the ability to produce a signal transformation into an orthogonal space, but the signal must be periodic. Additionally, the requirement that the signal must be stationary makes it difficult if we are to represent time-varying signals. To compensate for these problems, a technique called Discrete Windowed Fourier Transforms (Zayed [1993]) calculates the FFT over a window of discrete size at different temporal points in the signal. By calculating this windowed FFT, the instance of the signal can be assumed to be stationary and periodic with a period equal to the width of the window.

The most obvious method of applying Fourier series expansion to an $M \times N$ image

is to compute the two dimensional Fourier transform:

$$F(\omega_m, \omega_n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k,l] \exp^{-jmk2\pi/M} \exp^{-jnk2\pi/N}$$

where $f[k,l]$ is the pixel value in the image a location (k,l), $\omega_m = m\frac{2\pi}{M}$ and $\omega_n = n\frac{2\pi}{N}$ are the frequency components in the X and Y directions, respectively. Additionally, this series can be evaluated in a number of directions by rotation of the axes.

Other methods exist in which a two-dimensional image is represented as a one dimensional signal. Elms [1994] has applied Fourier series expansion to the recognition of machine printed characters by using an HMM with a single pixel wide sliding window in which the height of the grey level profile is taken to be the function, $f(t)$, where $t = 0$ is defined as the top of the window, and t increases scanning from the top to the bottom of the window. An FFT is used to transform this to the frequency domain, producing the magnitude spectrum for that 64-pixel tall window. The resulting 64 term spectrum is reduced to the first 32 terms. These, along with the center of gravity of the profile, are used as a 33-dimensional feature vector.

Fourier Descriptors

Another way of representing an image pattern as a signal is by tracing the contour of an object in the image. One technique (Zahn and Roskies [1972]) uses the *cumulative angular function* (CAF) created by one revolution of tracing to define a windowed signal which is expanded using Fourier series expansion. Another technique (Granlund [1972]) represents the contour as a non-windowed periodic function by calculating the function:

$$u(s) = x(s) + jy(s),$$

where s is a measure of the distance traveled around the contour. If the length of the entire contour is S, then this function is periodic with period S.

Moment Fourier Descriptors

A more recent moment invariant that has been proposed is the *moment Fourier descriptor* (Wang, Chen and Lin [1994]). This is a technique of calculating the Fourier coefficients for a number of line segments found by searching in a number of directions from the centroid of an object in the image. The calculated coefficients have the ability to represent an arbitrarily complex object while remaining invariant to rotation, translation, and scale.

The object is divided into N distinct vectors which start at the centroid of the object and point outward at equally spaced angles. For each of these vectors, a number of areas of blackness are encountered. The moment of the intersection of the object with the kth vector, r_k, is then calculated as:

$$M_k = \sum_{m=0}^{N_k-1} |o_{km} - o| L_{km},$$

where N_k is the number of black areas encountered by vector r_k, o is the centroid of the object, L_{km} is the length of the line segment created by the intersection of r_k and black area m, and o_{km} is the centroid of this line. Taking M_k as a periodic function with period N, the coefficients of its Fourier series expansion can be calculated. One important difference between this use of the Fourier series and the windowed Fourier series approach is that the moment Fourier descriptor requires that the patterns to be recognized must be presegmented.

2.3.3 The Gabor Transform

The Gabor transform is a variation of the Windowed Fourier Transform. In this case, the window used is not a discrete size, but is defined by a Gaussian function (see Zayed [1993, Ch. 10]). A generalization of this type of transform can be written as:

$$G(f, \omega, t) = \int_{-\infty}^{\infty} f(x) g(x - t) \exp^{2\pi i \omega x} \, dx$$

where $g(x - t)$ is the window function centered at time t. In the discrete windowed Fourier transform case, this is a window defined as $g(x) = 1$ for every $t - \epsilon \leq x \leq t + \epsilon$, and $g(x) = 0$ elsewhere. For the Gabor transform, $g(x)$ is described by the Gaussian probability density function.

The application of Gabor transforms to OCR has recently been explored in the recognition of Chinese characters (Hamamoto et al. [1995]). More commonly, they have been used with good success in segmenting textual regions in document images (Jain and Bhattacharjee [1992]). These methods use the two-dimensional version of the Gabor transform.

2.3.4 Wavelets

Details of an image at some resolution may have very different characteristics than other resolutions. Therefore, it may be to our advantage to handle them separately. Wavelets are a technique of series expansion that allows us to represent the signal at different resolutions. The series is created using a number of orthogonal functions that are constructed from a single function, known as the *mother wavelet*, by translation and dilation. This produces a series of functions that can represent the signal at different resolutions.

Let $\Psi(x)$ denote the mother wavelet, then the wavelet function at resolution 2^j

can be calculated as

$$\Psi_{2j}(x) = 2^j \Psi(2^j x)$$

Furthermore, the mother wavelet is defined in terms of some scaling function, $\theta(x)$. One common scaling function is defined as follows:

$$\theta(x) = \begin{cases} 1 & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

This can be used in defining the *Haar wavelet*:

$$W_H(x) = \theta(2x) - \theta(2x - 1).$$

This type of wavelet has been applied to handwritten numeral recognition (Lee and Kim [1995]) using a technique of applying wavelets to images developed by Mallat [1989]. In two dimensions, the Haar wavelet can be applied in different directions. At the resolution 2^{-1}, the image is represented by every other pixel from the original image. The low frequency and high frequency components at this resolution can be measured by convolving the image with each of the four 2×2 masks shown in Figure 2.4. This creates four new images, one for each mask. Furthermore, the image containing the low frequency components at the resolution 2^{-1}, can be broken into high frequency and low frequency components at the resolution 2^{-2} by convolution with each of the masks at this resolution. In Figure 2.4, which reproduces Figure 2 by Lee and Kim [1995], mask (a) is used to obtain the low frequency components, while masks (b), (c), and (d) are used to obtain the horizontal, vertical, and diagonal high frequency components, respectively. This process can be continued to any arbitrary resolution, creating a representation of the image at multiple resolutions.

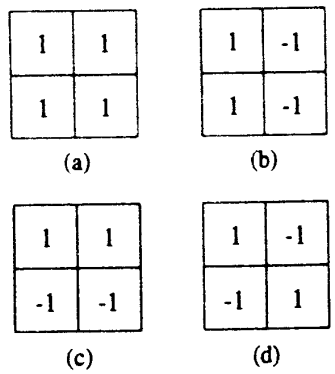

Figure 2.4. Two-Dimensional Haar Wavelet Masks

2.3.5 Karhunen-Loeve Expansion

Using the *Karhunen-Loeve* expansion, a transformed series is constructed as a linear combination of the original features such that the resulting features are uncorrelated. This can be accomplished using a transformation that rotates the coordinate axes so that the resulting covariance matrix of the data becomes a diagonal matrix. For n sample vectors $\mathbf{x}_i$ with dimensionality d, we have the $n \times d$ sample matrix, $\mathbf{A}$, with $d \times d$ covariance matrix $\mathbf{R}$. There exists a transformation matrix, $\mathbf{C}_R$, that transforms the sample data:

$$\mathbf{B} = \mathbf{A}\mathbf{C}_R^T$$

such that $\mathbf{C}_R$ is an orthonormal basis:

$$\mathbf{C}_R^T\mathbf{C}_R = \mathbf{I},$$

where $\mathbf{I}$ is the $d \times d$ identity matrix, and the covariance matrix of the transformed data is diagonal:

$$\frac{1}{n}\mathbf{B}^T\mathbf{B} = \mathbf{C}_R\mathbf{R}\mathbf{C}_R^T = \mathbf{\Lambda}_R$$

where $\mathbf{\Lambda}_R = diag(\lambda_1, \ldots, \lambda_d)$. The parameters, $\lambda_i, 1 \leq i \leq d$ are the *eigenvalues*, and the vectors that make up the rows of $\mathbf{C}_R$, the eigenvectors of the covariance matrix for $\mathbf{A}$.

The eigenvalues measure the scatter of the data in the transformed space, and therefore represent the information that is provided by each dimension of the transformed vectors. Since this is merely an linear transformation, and therefore no information is lost, these eigenvalues correspond to the amount of total information contained in the original features that is encoded in each dimension of the new features. A reduction in feature dimensionality can be achieved by eliminating those feature dimensions which contribute little information. This reduces the data to m-dimensional vectors, $1 \leq m \leq d$, such that

$$\frac{\sum_{i=1}^{m} \lambda_i}{\sum_{j=1}^{d} \lambda_j}$$

measures the fraction of total information in the original data representation that remains in the reduced representation.

This method of data reduction is also known as *principal component analysis* since it projects the data onto the principal components of the covariance matrix. It can be shown (Fukunaga [1990]) that this method of dimension reduction is optimal in terms of minimizing the square error of the data approximation.

2.4 Delta Features

A continuous signal can be represented as a sequence of amplitudes taken at discrete time intervals. This type of representation carries with it a lot of redundant information if amplitudes do not change rapidly from interval to interval. A more compact encoding of the signal may be to measure the amount of change in the amplitude from one interval to the next. This can also be applied to other types of continuous value features. In addition to a reduction in dimensionality, this type of encoding can be useful for providing some information about the past state of the signal. This will be further discussed in Section 5.5.2.

2.5 Summary

Good features should represent parts of the signal that are important to classification, while reducing the amount of redundant and unnecessary information that is stored in the representation. Three different groups of feature extraction techniques have been introduced: templates, structural decomposition, and series expansion, and their representation as delta features was discussed as well. Templates have the problem that they are very susceptible to small variations, and therefore only have limited application. One successful method applies a number of small templates to the image and counts the number of matches in different regions. Structural decomposition strives to represent structural properties of an object. This may take the form of an encoding of the object's contour or a search for components such as T-joints formed by the written strokes of a word. Series expansion is a method of representing a signal as a series of coefficients created by projecting the signal onto some basis. This has the advantage of allowing us to eliminate coefficients that contribute very little to the signal, with only a small amount of error imposed on the reconstructed signal. The Karhunen-Loeve expansion is one such expansion that is guaranteed to

minimize reconstruction error. Delta features are a method of reducing the amount of information that must be stored for a discrete time representation of a signal by storing only a value that measures the amount of change that has occurred in the signal. As such, it can be applied to any feature extraction method. However, the introduction of deltas can be expected to yield improvements only if the feature sequences are semi-stationary.

CHAPTER 3

Hidden Markov Models

Hidden Markov Models (HMMs), have been very successful in the field of Speech Recognition over the years due to their ability to model the passage of time. More recently, the benefits of their application to different forms of Optical Character Recognition (OCR) have been shown. In particular, one problem that has proved troublesome using traditional methods of recognition is that of cursive handwriting recognition. On-line handwriting recognition, in which the temporal information of how characters or words are constructed by the writer is available, is a clear candidate for the use of an HMM approach due to its similarity to speech signals. In addition, methods of simulating a temporal signal from static data have been used to apply Markov recognition techniques to off-line handwriting and have shown promising results in the recognition of cursive handwriting. This chapter shall present an introduction to the theories and algorithms behind Hidden Markov Models, as well as the issues involved in their use in practical applications. The treatment here is greatly indebted to Rabiner [1989] and Rabiner and Juang [1986].

3.1 Introduction

The Markov Model is a method of modeling a signal as a sequence of observable outputs produced by some process called the *source*. Modeling a signal this way allows us to potentially learn about the source that created the signal without having to directly examine the source itself (Rabiner [1989]). With such a model we have the ability to recognize a signal that was produced by the source, or to predict the most probable future observations of the signal given a partial sequence of observations. The Markov Model makes the assumption that the signal was produced by a Markov source - a type of source in which symbols currently produced are dependent only on a fixed number of symbols that have been produced preceding the current output. The *order* of a model specifies the number of preceding outputs which are taken into account for the next symbol to be produced. Since the complexity of the model grows exponentially with the order, and the added benefit of increasing the order of the model decreases as the order grows higher, first-order or second-order Markov Models are accepted as sufficient for most applications.

The outputs corresponding to a single symbol can be characterized as discrete or continuous. Discrete outputs may be such things as characters from a finite alphabet or quantized vectors from a codebook, while continuous outputs might be speech samples or music which are represented as samples from a continuous waveform.

In general, a signal can be classified in two categories: those whose statistical properties vary with time, and those whose statistical properties do not vary with time. These have been referred to as time-varying (or nonstationary) and time-invariant (or stationary), respectively (Rabiner and Juang [1986]). A time-invariant signal could be a sine wave whose parameters (amplitude, frequency, and phase) remain constant over time, while an example of a time-varying signal could be a sound wave, which can be characterized by a combination of sine waves whose frequencies, amplitudes,

and phases change over time. A popular Markov Model approach represents a time-varying signal as a number of short time-invariant segments. These segments are what make up the output sequence (often referred to as the observation sequence) that represents the signal.

A signal model may exploit some known properties of the signal, or it may only rely on statistical properties of the signal. If the known properties of the signal are used to benefit the model, the model is known as deterministic, and all that need be done is to estimate the parameters of the model (such as if we know the signal is some form of sine wave). A model that relies only on the signal's statistical properties is known as a statistical, or stochastic, model. In this case it is assumed that the signal can be characterized as a random process. The Markov Model is a type of stochastic model.

The Hidden Markov Model (HMM) is a doubly stochastic variant of the Markov Model, with an underlying stochastic process that is not observable (hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols (Rabiner and Juang [1986]). The underlying process of the HMM is a finite automaton (see Figure 3.1 which reproduces Figure 2 by Elms [1994]). It contains states, state transitions, and transition probabilities. Initial state probabilities exist for each state which define the chances of the model being found in that particular state at the beginning of an observation sequence. Observation densities, which indicate the probability of a particular observation being produced by the model, are defined for each state, and these make up the observable stochastic process. In a given state, the observation probability density function for that state defines the expectancy of observations in that state, while the sequence of states traveled define a movement from one expectancy of observations to another. It is unknown to an outside observer what state the source (or the model representing the source) is in at any given time. All that can be observed is the sequence of symbols

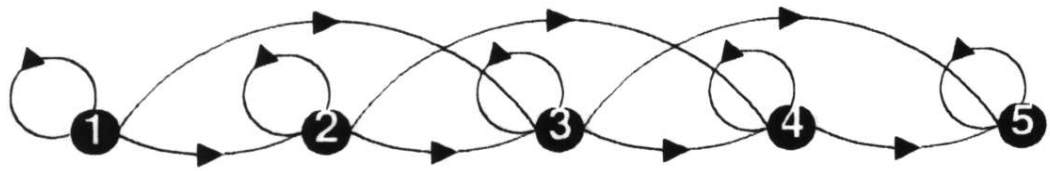

Figure 3.1. A Simple Left-Right Hidden Markov Model

produced that make up the signal. Therefore, this type of model is referred to as Hidden.

A signal is produced by a Hidden Markov Model by first selecting a starting state for which the model will be considered to exist at the time in which the signal originates. For simplification, this can be thought of as time $t = 1$, although it may be the case that previous signals produced (or an earlier portion of the present signal) may have helped determine what state the model is presently in. At this time, some symbol is produced which is the first observation of the output signal. In deciding which symbol will be produced, the model will randomly select a symbol from a set of all possible symbols that can be produced by this state, with probability determined by the observation density function for that state of the model. At the next time segment, time $t = 2$, the model will move to a next state by selecting a state transition from the set of all possible transitions for the present state with probability determined by the present state's transition probability density function. Having arrived in this new state, the new state will select a symbol to be produced from its possible symbols with probability determined by this new state's observation density function. The model continues in this fashion where each tick of the clock (increment in time, t) brings another state transition followed by the emission of another symbol. This sequence of emitted symbols defines a discretized version of a signal that could

have been produced by the real source (assuming the model accurately represents the source).

Since the probability of the state sequence is equal to the product of the probability of the initial state and the probabilities of all transitions that were taken multiplied together, and the probabilities of all symbols that were emitted in each of these states are known, we can easily calculate the probability of this state sequence occurring and this symbol sequence being produced by multiplying these values together. This is not the case, however, when trying to determine the probability that a symbol sequence was produced by a model when the state sequence is unknown. Such is the case when using Hidden Markov Models for recognition of signals produced by a source where we have no information on how these signals were produced by the source and therefore it may be the case that many different state sequences could be a valid choice to produce this signal. This will be the topic of Section 3.3.

3.2 Model Representation

As was described in the previous section, each state in the model has a number of parameters associated with it which describe the probability of making a transition from that state to another state in the model (possibly itself), the probability of particular observations being produced while in that state, as well as the probability that a particular state was the start state for the sequence under observation. We now define a notation based on that given in Rabiner [1989] which will assist in the discussion of these models.

Define:

$O = \{O_1, O_2, \ldots, O_T\}$ as the observation sequence

$T =$ the length of the observation sequence

$Q = \{q_1, q_2, \ldots, q_N\}$ as the set of states in the model

N = the number of states in the model.

In addition, for discrete HMMs:

$V = \{v_1, v_2, \ldots, v_M\}$ is the set of possible observations

M = the number of observation symbols.

For continuous HMMs, the number of different possible observation symbols is infinite.

In the discrete case, a Hidden Markov Model can be described by a set of parameters $\lambda = (\mathbf{A}, \mathbf{B}, \mathbf{\Pi})$, where $\mathbf{A}$ and $\mathbf{B}$ are matrices, and $\mathbf{\Pi}$ is a vector. The differences for the continuous case will be discussed in Section 3.6.2.

Matrix $\mathbf{A}$ is defined as the set of all state transition probability distributions for the model:

$$a_{ij} = P\{q_j \text{ at } t+1 | q_i \text{ at } t\},$$

where q_i is the state the model is in at some time t, q_j is a possible next state, and a_{ij} is the probability that the model will make the transition to this next state at the next time increment.

Matrix $\mathbf{B}$ is defined as the set of all observation probability distribution functions for the model:

$$b_j(k) = P\{v_k \text{ at } t | q_j \text{ at } t\},$$

where q_j is the state the model is in at some time t, v_k is an observation that may be seen by the model, and $b_j(k)$ is the probability of this observation occurring in the observation sequence while in state q_j.

Vector $\mathbf{\Pi}$ is defined as the initial state distribution for the model:

$$\pi_i = P\{q_i \text{ at } t = 1\},$$

where q_i is a state in the model and π_i is the probability that the model will begin in state q_i at the beginning of an observation sequence.

3.3 Model Evaluation

So far we have viewed a signal (a sequence of symbols) as being produced by a model but when an HMM is used for the purpose of classification, we view a sequence of symbols as having been produced by unidentified sources. The goal of classification is then to decide which source this observed sequence was produced by. In this case a number of HMMs exist, one to model each respective source. The sequence of symbols is presented to each model and its probability of being produced by that model is evaluated. The model that has the greatest likelihood of producing this observation sequence then defines the chosen identity of the source which is believed to have produced it.

In order to evaluate models, we must first calculate $P(O|\lambda)$, the likelihood that a sequence of observations, O, were produced by a given model, λ. For any observation sequence produced by a model, and a known sequence of states followed to produce this sequence of observations, the probability of the model producing these observations and following this state sequence can be calculated as:

$$P(O, s|\lambda) = \pi_{s_1} b_{s_1}(O_1) \prod_{i=2}^{T} a_{s_{i-1}s_i} b_{s_i}(O_i).$$

In most practical situations, however, the state sequence followed is unknown to us. Since there may be many state sequence paths that can produce this observation sequence, the probability that O was produced by model λ is the sum of all the probabilities $P(O, s|\lambda)$ for all possible state sequence paths, s. Computing the probabilities for each of these paths would be computationally very expensive, requiring on the order of $2TN^T$ calculations. Fortunately, an efficient algorithm using a technique called *forward probabilities* exists which can calculate this sum of probabilities in on the order of N^2T calculations, and additionally can report the state sequence

of the most probable path. This is known as the *Forward-Backward algorithm* (Baum [1972]) since it can be similarly defined using *backward probabilities.*

Define $\alpha_t(i)$, the forward probability, as the probability of the partial observation sequence from time 1 to time t occurring, and the model being in state q_i at time t. This can be solved for inductively:

1. $\alpha_1(i) = \pi_i b_i(O_1),\ 1 \leq i \leq N$

2. For all $t = 1, 2, \ldots, T-1$:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}), 1 \leq j \leq N$$

The probability of the complete observation sequence, O, can then be solved:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

In a similar fashion, the backward probabilities, $\beta_t(i)$, can be defined as the probability of the partial observation sequence from time t+1 to time T occurring, and the model being in state q_i at time t.

1. $\beta_T(i) = 1,\ 1 \leq i \leq N$

2. For all $t = T-1, T-2, \ldots, 1$:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}(j)), l \leq j \leq N$$

This will be useful later. The underlying principle of the forward-backward algorithm is the following: if we know the probabilities, for each state q_i, of the partial sequence of observations up to time t leaving the model in state q_i, the probability

of the model being in some state q_j at time $t+1$ can be calculated using only those probabilities, and those for times before t are not needed. Therefore, the forward probabilities act as accumulators for each state as we increase t from 1 to T, and the value at time T is the probability of being in that state at the end of the observation sequence.

The forward and backward probabilities have been found to be useful for other computations involving state path traversals. This will be shown in algorithms used to reestimate model parameters presented in the next section.

With the help of the forward and backward probabilities, we now have the probability that an observation sequence was produced by a given model. In order to be able to choose one competing model over another we must be able to compare, for each model, the probability that given the observed sequence, the model produced it. This can be calculated using the Bayes Rule (see Section 1.6.2).

3.4 Estimating Model Parameters

In creating a model of a source, the details of the source's inner workings are mostly unknown. We need to be able to define the characteristics of our model based on previous observation sequences, or *training examples*. Therefore, we require a method of estimating the model parameters using these examples. The most common method of estimating a model's parameters is to set them so as to maximize the probability of the training observation sequences being produced by the model (Rabiner [1989]). This is a maximum likelihood method of training. Since an analytical solution to estimating the parameters is intractable, the parameters must be estimated iteratively or by some method of gradient descent. A very common iterative re-estimation method is the Baum-Welch algorithm:

First define

$$\gamma_t(i) = P(q_i \ at \ t|O, \lambda)$$

and

$$\xi_t(i,j) = P(q_i \ at \ t, q_j \ at \ t+1|O, \lambda)$$

These are respectively the probability of the model λ being in state q_i at time t when presented with the sequence of observations O, and the probability of the model λ being in state q_i at time t and making a transition to state q_j at time $t+1$ when presented with the sequence of observations O. These can be calculated using forward and backward probabilities as follows:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

Model parameter estimates can now be updated as follows:

$$\overline{\pi}_i = \gamma_1(i)$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\overline{b}_j(k) = \frac{\sum_{\substack{t=1 \\ O_t=k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Each iteration of the Baum-Welch algorithm has been proven to increase $P(O|\lambda)$

for that instance of O until a local maximum has been reached (Baum [1972]).

The Baum-Welch algorithm is by far the most well-known parameter re-estimation technique in use for HMMs. This does not mean, however, that it is the only technique, or even necessarily the optimal technique for re-estimation. Gradient descent techniques have also been applied with good results (Levinson, Rabiner and Sondhi [1983]). Levinson, Rabiner and Sondhi [1983] also discuss the use of Lagrange multipliers for parameter estimation. In addition, it should be pointed out that all these estimations are maximum-likelihood estimators since they focus on maximizing an individual model's likelihood of having produced observations from the source that it is modeling, but does not take into consideration the likelihood of competing models producing these same observations. This may produce sub-optimal decision boundaries. A better estimator would be one that is based on Maximum Mutual Information (MMI) (Bahl et al. [1986]). This method focuses on optimizing the parameters of the set of models simultaneously, making adjustments so that an observation sequence O_v, that was produced by a source v modeled by model λ_v, has a higher probability of being produced by λ_v than any model other than v. Unfortunately, the amount of computation required makes MMI training infeasible for most HMM applications. Experiments in which simplifying assumptions have been made in order to reduce the required computation have been conducted on the use of MMI training and have shown that significant improvements over maximum likelihood training can be achieved. In one example, Bahl et al. [1986] measure the mutual information between a model and a sequence of observations by the following equation:

$$I_\theta(\lambda_i, O) = \log P_\theta(O|\lambda_i) - \log \sum_{j \neq i} P_\theta(O|\lambda_j) P(\lambda_j),$$

where $I_\theta(\lambda_i, O)$ is the mutual information between the observation sequence O and a model, λ_i, from the set of models, θ, each of which characterizes a particular source

class. The goal of training is to create the models in θ so as to maximize the value of $I_\theta(\lambda_i, O)$, and therefore the mutual information. A similar method, known as Corrective Training (Bahl et al. [1988]) focuses on minimizing recognition errors by iterative parameter readjustment and has been shown to give better results than Maximum Likelihood Estimation. At each iteration, every incorrectly classified signal element causes the model which was incorrectly chosen to update its parameters so as to decrease that element's probability of being produced by that model. Similarly, the model that should have recognized this observation sequence will have its parameters adjusted so as to increase that sequence's likelihood of being produced by the model. This continues until the models converge to a point where no training observations are found in error. Unfortunately, although Corrective Training can produce better results than maximum likelihood training, it is still computationally more expensive and is not guaranteed to converge.

Another alternative to the Maximum Likelihood estimator is based on Minimum Discrimination Information (MDI) (Ephraim, Dembo and Rabiner [1987]). This attempts to minimize the discrimination information between the probability densities of the HMM and the true densities of the source being modeled. Unfortunately, obtaining such an minimum is highly non-trivial and therefore training methods based on this approach are scarce.

3.5 Picking an Optimal State Sequence

Given a model and a sequence of observations, one problem that needs to be addressed is the selection of an optimal sequence of states traversed to create this observation sequence. It is more than likely the case that many possible state sequences could produce a given observation sequence. One way of selecting a state sequence is to, for each observation of a sequence, choose those states that have the highest individual

probability of producing that observation. The problem with this method is that, having not taken into account the actual state connectivity, an impossible state sequence might be selected. A method of selecting the best path over state transitions is required. A well known efficient method is a dynamic programming algorithm known as the *Viterbi Algorithm* (Viterbi [1967]; Forney [1973]). The algorithm obtains a solution recursively as follows:

Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N$$

$$\Psi_1(i) = 0$$

Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(j) a_{ij}] b_j(O_t), 2 \leq t \leq T, 1 \leq j \leq N$$

$$\Psi_t(j) = \arg \max_{1 \leq j \leq N} [\delta_{t-1}(i) a_{ij}], 2 \leq t \leq T, 1 \leq j \leq N$$

Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$

$$i_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

Path Backtracking:

$$i_t^* = \Psi_{t+1}(i_{t+1}^*), t = T-1, T-2, \ldots, 1$$

The Viterbi algorithm is very similar to the Forward-Backward algorithm and, in fact, could be used in place of the Forward-Backward algorithm to evaluate a model, given an observation sequence. The Viterbi Algorithm contains a parallel to

the Forward variable $\alpha_t(i)$, denoted as $\delta_t(i)$, however while $\alpha_t(i)$ represents the sum of probabilities of all possible paths into a state q_i at time t, $\delta_t(i)$ is the maximum of these different path probabilities. $\delta_T(i)$ is then the probability that the maximum probability path ends in state i, and P^* is the total maximum path probability of all paths ending in any on the N states. Similarly, i_t^* is this final state that the sequence will end in with probability P^*. Therefore, by starting at state i_T^* and backtracking using $\Psi_t(i)$ to reach state i_{t-1}^* from any state in the path i_t^*, we can completely reconstruct this path.

3.6 Observation Densities

Each state of an HMM has an observation density function, $b_j(O)$, that describes the probability that a given observation O was created by the model while in this state. Since usually neither the true densities nor reliable approximations of these densities are known in advance, an approximation must be estimated using the training data. Three main types of observation probability densities can be found in the literature that are used in HMMs. *Discrete densities* assume a finite variety of observation vectors exist, such that the probability of each occurrence can be estimated. *Continuous densities*, on the other hand, represent the output distributions by a continuous distribution which is often estimated as a mixture of Gaussian densities. *Semi-continuous densities* (also known as *tied mixture models*) are a technique of creating observation densities as a weighted combination of Gaussian prototypes created by a clustering of the data independent of its class.

3.6.1 Discrete Densities and Vector Quantization

Discrete density HMMs represent every observation as one of a number of discrete vector prototypes. By observing the frequency of each of these prototypes, the probability of each can be estimated. In their original form, the observations may take the same form as these prototype vectors, take on a larger variety of discrete values, or may take on values from some continuous distribution. In either of the last two cases, before training a model we must first create a *codebook* of prototype vectors using *vector quantization* (Gray [1990]) so that any observation can be represented by one of these prototypes. When creating this codebook, a predetermined number of clusters must be created that will partition the input space into linearly separable regions, one for each codebook vector. This partitioning is most often done by selecting an initial set of prototype vectors and iteratively improving on them.

Gray [1990] describes methods of selecting an initial codebook. Methods of random selection include selecting the first set of observations from the training data to act as the initial codebook vectors, and selecting training observations at regular intervals to act as these initial vectors, the latter of which acts as a better starting point if data are highly correlated.

Once an initial set of codebook vectors has been selected, training observations are clustered by classifying each to its closest prototype vector. Using these new clusters, each codebook vector can be recalculated to more accurately reflect the real cluster of vectors that are falling within its cluster. It has been shown (see Gray [1990]) that the best vector quantized representation of a cluster of vectors is that cluster's centroid. With this new codebook, iterations of classifying the training data and re-adjusting the codebook vectors are continued until the average distortion induced by this representation falls below some threshold level. Often a maximum number of iterations is specified for this algorithm to remedy the case where the average

distortion below a certain threshold cannot be obtained.

Clusters can be created using data from the entire set of training observations as a single pool of data, or by using class labels, provided with the training observations, to create clusters from each individual class pool of data. The former technique is known as intrinsic, or unsupervised, clustering while the latter is known as extrinsic, or supervised, clustering (Jain and Dubes [1988]). As an example, these labels may be characters from an alphabet, words, phonemes, or some smaller sub-grouping of one of these. Labels are often provided by humans through a very tedious job called truthing. Due to the large number of man-hours that this job can require (which grows proportional to the size of the training data pool) it may be the case that higher level labels (such as word labels) will be available, but labels of higher granularity, such as characters, are not available. In this case a supervised clustering can be performed at the word level, while unsupervised clustering must be performed at the sub-word level.

When vector quantization is used to create a finite number of discrete prototypes to represent continuous data, this representation becomes a compressed version of the original data. Associated with this compression will be a quantization error. In order to construct good vector quantized codebooks, we need a measure of the distortion imposed on the data by this representation. One popular distortion measure used in vector quantization is the Mean Squared Error (MSE). This is basically the sum of squared Euclidean distances between each observation vector and the quantized vector representing it:

$$d(O, \hat{O}) = \frac{1}{T} \sum_{t=0}^{T-1} d(O_t, \hat{O}_t),$$

where

$$d(O_t, \hat{O}_t) = \|O_t - \hat{O}_t\|^2.$$

3.6.2 Continuous Densities

As opposed to discrete densities, continuous densities (Bahl et al. [1990]) represent the observation densities as Gaussian distributions estimated for each state's data. Since the state alignment of the data will change as the model is trained, these Gaussian distributions must be iteratively estimated during model training. Multiple densities, often referred to in the literature as *mixture models* (Baker [1990]), may be used to represent a single state's data whose distribution is more complex than a single Gaussian can represent. These Gaussians can be combined as a weighted sum of each Gaussian:

$$b_j(\hat{O}) = \sum_{m=1}^{M} c_{jm} \Pi[\hat{O}, \hat{\mu}_{jm}, \hat{U}_{jm}],$$

where j is the state number, m is the mixture number, Π is the Gaussian density function with mean $\hat{\mu}_{jm}$ and covariance matrix $\hat{U}_{jm}$, and the coefficient c_{jm} is the contribution (weight) of that Gaussian to the overall mixture. Certain constraints that must be imposed upon these parameters as well as the re-estimating equations are presented (Rabiner [1989]):

Constraints:

1. $\sum_{m=1}^{M} C_{jm} = 1$, for all states j

2. $c_{jm} \geq 0$ (no Gaussian can have a negative contribution to the combination)

3. $\int_{-\infty}^{\infty} b_j(\hat{x}) d\hat{x} = 1$

Re-estimation:

$$\hat{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{m} \gamma_t(j,k)}$$

$$\hat{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) O_t}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

$$\hat{U}_{jk} = \frac{\gamma_t(j,k)(O_t - \mu_{jk})(O_t - \mu_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)},$$

where

$$\gamma_t(j,k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \frac{c_{jk}\Pi(O_t, \mu_{jk}, U_{jk})}{\sum_{m=1}^{M} c_{jm}\Pi(O_t, \mu_{jm}, U_{jm})}$$

is the probability of being in state j at time t with the kth mixture component accounting for O_t. Note that by setting $M = 1$, we have the re-estimation algorithm for the single mixture density case.

Diagonal Covariance vs. Full Covariance

The covariance of a Gaussian can be represented in one of two ways. The first assumes that each dimension of the Gaussian (each feature) is statistically independent of all the others, so that the value of one feature remains unaffected by a change in the value of any other feature. The second representation makes no such assumption and represents the covariances as the set of independent variances, plus all covariances between each pair of features. For a d-dimensional Gaussian, this creates a $d \times d$ matrix of covariances, called a *full covariance matrix*, with the main diagonal containing the independent variances. A Gaussian variance represented only by a set of independent variances is then called a *diagonal covariance* since it consist of a covariance matrix in which all components are zero accept the components on the main diagonal.

If a set of features are not independent of each other, as is often the case, then a full covariance model should be more capable of modeling the data than a diagonal covariance model. However, it is not always the case that a full covariance model can be adequately trained. For a Gaussian model of a d-dimensional data set, the diagonal covariance contains d parameters that must be estimated. On the other hand, the

full covariance requires $\frac{d \times (d+1)}{2}$ parameters to be estimated, placing a much greater burden on the data used for training (note that the covariance matrix is symmetric, therefore the redundant entries can be eliminated). We will see examples of diagonal versus full covariance models in Section 4.6.1.

3.6.3 Semi-Continuous Densities

The need for more data to facilitate the training of more powerful models has led to the use of *tied mixtures* (Euler et al. [1990]). The concept behind tied mixture models is that similarities may exist in the data between observations that do not represent the same source. This implies that clustering the data in an unsupervised fashion should create clusters of data which cross class boundaries. By estimating Gaussian distributions for these clusters, we can represent any state's distribution as a weighted combination of Gaussian prototypes known as *semi-continuous densities* (Huang and Jack [1990]). Since this method trains Gaussians on areas of the input space where there is a large amount of data, this method has the effect of applying the available model parameters where they are most needed to represent the input space. The danger of this method is that it focuses on clustering together similar observations, and not observations that provide discriminating information.

Kaltenmeier et al. [1993] describe their system in which observations are clustered into one of a number of Gaussian prototypes in a tied-mixture model. They are created by clustering the training observations in an unsupervised fashion, disregarding their origin. Models are then created in which observation densities for each state can be made up of some combination of any of the Gaussians regardless of whether or not observations that were used to create some of those Gaussians came from that particular element being modeled. In this case, an observation distribution for a state in a model of the letter *d* might contain a prototype that was created in part by observations of the letter *o* (i.e., the start of the loop). Tied-mixture models have

become popular in recent years due to the availability of large data pools which can be used in clustering from all training data regardless of what class it came from. Since truthing is a very labor-intensive process, any technique that can make use of untruthed data is of great practical significance. By clustering together similar observations from different classes, the amount of data available to create the codebook vector representation for that cluster is greater, and more powerful models can be trained. Unfortunately, by estimating a class observation density using observations from other classes, the potential is increased for one model to confuse observations from another class as observations from its class.

To illustrate the difference between continuous and semi-continuous models, we describe here how a very similar method can be used to create a continuous density model. Untied-mixture models can be constructed with prototypes created by a supervised clustering of the training data. This involves forming clusters in which labels on the training data are used to force differently labeled training data to fall into disparate clusters. Only vectors used to represent observations from one class will be used in creating the model for that class, and not from any other classes. While this type of model lacks the advantages of the tied models mentioned above, these advantages are not the same as maximizing the discrimination between different models. We will see examples of semi-continuous models versus continuous models in Section 4.6.3.

3.7 Lack of Training Data

Another problem that is encountered during model parameter re-estimation, due to a finite size training set, is that which occurs when a model parameter is set to zero. If, for instance, after training the model, the probability of seeing an observation, v_k, in state q_j is set to zero (i.e. $b_j(k) = 0$). The true distribution of the source from

which the training observations were produced may be such that it is possible for a signal to be produced in which v_k is observed in state q_j, however since no such case ever occurred in the training data, this model will never recognize such a signal since $b_j(k) = 0$ states that this is an impossible observation. This problem can easily be overcome by restricting all parameters from falling below some minimum value. Modifications of the Baum-Welch algorithm have been formulated to take into account this new constraint (Levinson, Rabiner and Sondhi [1983]). In our experiments, a minimum value of 0.05 for all variances on the diagonal of the covariance matrix was enforced.

Lack of training data can often result in parameters receiving poor values and certain characteristics of the source being poorly modeled or even left out all together. This may possibly be remedied by reducing the size of the model, thus reducing the number of parameters that need to be estimated. Unfortunately, if the structure of the model was chosen because it is believed to accommodate some inherent characteristics of the source, changing this is undesirable. Jelinek and Mercer [1980] have devised a method of interpolating the parameters of a model with the parameters of a reduced model for which there is adequate training data. In our experiments, categories for which there was not enough data to train an individual model were combined into meta-categories. This does not cause a problem if the goal of the system does not involve discriminating between the categories within these meta-categories. Examples of this technique will be seen throughout Chapter 5.

3.8 Initial Parameter Values

During the training of a model, model parameters are estimated using training examples as described previously. Both iterative methods of estimating these parameters, such as the Baum-Welch algorithm, and gradient descent methods, such as a neural

network, are based on creating a new set of parameters using results obtained from the model with a presently defined set of parameters. Therefore, some initial set of parameters must be picked in order for either of these methods of training to be possible. Since neither method of training guarantees an optimal solution due to the potential for getting stuck at a local minima, the choice of a good starting point can have an impact on how well the model will train.

Many of these initial parameter decisions must be made by a human. Parameters such as the number of states and the number of mixtures used to estimate the observation distributions will have a great impact on how each training example's influence is distributed across the model. For models with a small number of states, the observation density estimates for those states will be well trained due to the large number of examples that will go into their training. However, much of the contextual information that could be trained by spreading this data over more states may be lost. The final topology of the model is a factor that is greatly influenced by the initial values of the state transition matrix. If an infinite number of training examples were available, we could set all state transitions more or less equal, and let the data decide which transitions are most important. Since, in the real world, lack of data is always a problem, we can greatly help the model to focus on the important parameters by choosing which of these transitions should not be allowed, and which ones will probably be chosen over others. By setting the disallowed transitions to zero, and more probable transitions to higher values than the less probable transitions, the training process can concentrate on the important parameters. In our experiments, it was discovered that the initial values of parameters such as those that define the model topology, the number of Gaussian mixtures, and the dimensionality of the feature vectors have an impact on the effectiveness of the model. However, the exact initial value of many parameters, such as the initial values of the Gaussian means and covariances, and small changes to the initial values of the transition matrix, tend to

have little effect on the model after training.

3.8.1 Model Topologies

By placing certain restrictions on the structure of the model, the model will tend to behave differently. A model in which all states can be reached from any state is known as an ergodic model. For certain types of signals, other types of models have been shown to model that signal's properties better (Rabiner [1989]). In the case of a time-variant signal, a model that inherently represents the passage of time may be advantageous. The *left-right* model, also known as the Bakis model, is such a model. The underlying state sequence of this type of model is forced to start in state 1 (i.e., the leftmost state) and can only make transitions to higher states (i.e., to the right) or back to the present state as time passes. In addition, the model must end in state N (the rightmost state). This allows the sequence of states to represent the passage of time and has become the most popular form of Markov Model used in speech recognition and handwriting recognition. It is often the case that such models are simplified to only allow jumps of no more than two states, and this was the typical method of restricting jumps in our experiments.

Another form of state transition that has been found useful is known as the Null transition (Bahl, Jelinek and Mercer [1983]). The idea of the null transition is that a state transition should be allowed to occur without requiring an observation to be absorbed. What this amounts to is that null transitions are a special kind of state transition, with transition probabilities, that can occur without requiring the time counter, t, to be updated. The present observation in the sequence, O_t, will remain the present observation in the next state. A null transition from the end state of the model back to the start state will create a model capable of observing an arbitrary length string of elements which the model represents.

Due to self state transitions, a_{ii}, the Hidden Markov model can remain in the

same state throughout many transitions. For a given state, the number of transitions for which the model remains in the same state is known as the state's duration. This can be easily formulated as (Rabiner [1989]):

$$p_i(d) = a_{ii}^{d-1}(1 - a_{ii}),$$

where $p_i(d)$ is the probability the model being in state q_i, for d consecutive observations. This is an exponential distribution and is more than likely not the most appropriate model of the true duration density. One possible solution, in the left-right model, is to create a number of consecutive duplicate states to model the state duration. As an alternative, and a more accurate model of the duration, the self transitions can be eliminated and a duration density can be estimated for each state (Rabiner and Juang [1986]). Tracing an observation sequence through the model now requires not only tracing each possible path, but also must take into account all possible durations in each state. This same problem is encountered during model parameter estimation. A restriction must be placed on the model to limit the maximum duration to a predefined value, D, otherwise the number of durations that need to be checked would only be limited by the length of the observation sequence. Even with the restriction, the added complexity of modeling these state durations using this non-parametric approach is great, requiring about D times the storage and $D^2/2$ times the computation. Also, this method requires an additional $D - 1$ parameters per state (minus one because of loss of self transitions), and less data to estimate state transitions (since every state now claims multiple observations for $d > 1$). Parametric state duration densities, such as a Gaussian distribution or a gamma distribution of the duration, have been proposed as a simpler alternative duration estimators. For further discussion of duration modeling, the reader is referred to Kornai [1995, Ch. 3].

A great deal of effort was spent on finding appropriate topologies for these models. This will be discussed in Section 5.4.

3.8.2 Tied Parameters

Tying parameters involves recognizing that there are redundancies in the model that could utilize each others data when training. Parameters from two or more parts of a model can be considered merely pointers to the actual parameter values. During training, parameter updates for each of these parts of the model will have an influence on all other parts of the model for which these updated parameters are tied to. An example that we have already encountered in this writing is that of tied mixture models. However, the tied mixture models discussed previously had application only in semi-continuous densities, while this approach can be generalized to continuous densities as well, and can be more selective as to which parameters should be tied, and which should be independent. A rough example of this would be modeling a word in which the same letter occurs twice. Since both of these letters should come from the same approximate distribution, their state observation density estimates could be tied together so that when the parameters of one letter is changed, so are the others', effectively doubling the data used to train them. Our experiments used parameter tying only in the form of tied mixture models. This will be discussed in Section 4.6.3.

3.8.3 Multiple Feature Sets

If multiple independent feature sets are used by a classifier, it may be advantageous to create separate probability densities for the observations as represented by each of these feature sets rather than combine all these features into one long feature vector to be used in training a single set of densities. For each class, individual models may be trained each using one of these feature sets, and some combination of the

probabilities for those models (possibly a weighted sum) used to generate the final probability for that model class. Alternatively, these features sets can be combined into a single model by creating multiple observation density channels. Each channel contains, for each state, a set of Gaussians estimated using its own feature set, with the final observation density estimate for that state constructed as some combination of these channel densities.

Park and Lee [1993] describe three possible strategies for combining these probabilities. The simplest of these involves counting votes for the different feature sets. For each of the feature sets, models are scored and the highest scoring model defines the classification just as when a single feature set is used. This classification counts as one vote, and the class for which there is a majority of votes defines the final classification. A special case which must be handled occurs when there is no strong majority of votes. Therefore, the choice may be rejected if the majority vote count falls below some threshold (for Park and Lee [1993] this threshold was $K/2$, where K is the number of different channels). The second method involves averaging the probabilities of the models scored using different feature sets. However, it may be the case that the optimal combination of codebooks is some weighted combination. This third method can be expressed as follows:

$$P(O|\lambda_v) = \sum_{c=1}^{C} [w_c P(O_c|\lambda_{v_c})]$$

with the restrictions:

$$w_c \geq 0, c = 1, \ldots, C$$

$$\sum_{c=1}^{C} w_c = 1$$

where O_c is the sequence of observations expressed using the cth feature set, λ_{v_c} is the model created for class v using feature set c, and w_c is the weight of this model in the

combined model, λ_v, for class v. It was shown by Park and Lee [1993] that for their four different feature sets, combining feature sets by some empirically determined weighted sum clearly provided the best overall recognition, while the majority voting rule technique was consistently approximately 10% worst. One thing that was very clear in these results is that all of the combination techniques outperformed any individual codebook by a large margin ($> 20\%$). Experiments involving models trained using multiple feature sets evaluated as a weighted combination will be discussed in Section 5.5.8.

3.9 2D HMMs

The 2-dimensional nature of OCR suggests that there may be an advantage to modeling this type of data using a 2D HMM - a form of HMM in which a 2-dimensional matrix of states is used to model a 2-dimensional object within the image. Unfortunately, due to the large number of parameters involved, parameter estimation of such a model is extremely computationally expensive and requires large amounts of training data. Application of Pseudo 2D HMMs to printed character recognition have appeared in Kuo and Agazzi [1993], Agazzi and Kuo [1993a], and Agazzi and Kuo [1993b]. Unlike the fully connected true 2D models, these models are made up of a number of horizontally running left-to-right models, each represented as a superstate. These superstates then form a vertical top-to-bottom model.

None of our experiments have made use of 2D models, however some experiments were carried out in which the results of using a vertically sliding window were cross checked with that of a horizontally sliding window. By rejecting in cases where the two models do not agree, we were able to obtain a 91.84% recognition rate at 3.5% rejection on discrete presegmented characters (see Kornai, Mohiuddin and Connell [1995]).

3.10 Language Models

One of the added benefits of HMMs is that they have contextual processing built into them. This can be repeated at many levels (e.g., subword-level, word-level, sentence-level) by linking together individual models into some higher level model where the models at the lower level become states for the higher level (Bahl, Jelinek and Mercer [1983]).

N-gram probabilities, the probability of the joint occurrence of a sequence of N elements, provide a much better estimation than simple unigram prior probabilities. However, as N increases, the number of examples of N element sequences rapidly decreases so that many of these N-gram classes have zero occurrences in the training data. N-gram probability estimates based on the number of occurrences found in the training data would therefore be zero for many of the N-gram classes, while the true probabilities may be non-zero. Jelinek [1990] describes how a Markov model can be used to train the weights of a linear combination of the estimated trigram, bigram, and unigram probabilities. This forms a new trigram probability estimate which can take on a non-zero value even if no examples of that trigram class are present in the data:

$$P(w_3|w_1, w_2) = q_3 Freq(w_3|w_1, w_2) + q_2 Freq(w_3|w_2) + q_1 Freq(w_3).$$

One of the great advantages of using the Markov model as a language model arises due to problems in techniques of segmentation. In the OCR world, as well as in the speech recognition world, presegmenting words into smaller components, and then performing recognition on these components, is limited in that if these segmentation points are not good to begin with, the recognition module never stands a chance to find the correct character (Kornai [1994]). Oversegmentation can create extra segments known as insertion errors, while undersegmentation causes deletion errors.

In addition, misclassifications, known as substitution errors can occur on any of the segments.

3.11 Summary

Because character segmentation in cursive handwriting is a difficult task, it is desirable to perform segmentation during recognition. The Hidden Markov Model is particularly well-suited for such a task, as has been proven in the field of speech recognition. This chapter has presented methods of model representation, evaluation, and parameter estimation. HMMs model two types of stochastic processes: the sequence of states which the model travels through, and observation probability distributions for each state. The latter can take one of three forms: discrete, continuous, or semi-continuous. When representing the distributions as Gaussians, the effectiveness of full covariances versus diagonal covariances will be dependent on the true distribution of the data, and the number of examples available.

Many issues may arise when designing a model to represent a source. One of these is how to handle the case when there is not enough training data to train a good model. Decisions about the topology of the model and the use of parameter tying must be made. Additionally, there may be an advantage to using multiple feature sets and basing the final classification decision on a combination of the outcomes from the different feature sets.

Finally, higher level contextual processing can be provided by connecting models into a network where the individual models become states in a higher level HMM. If the lower level models represent words, the higher level model created defines a language model for all possible sentences.

CHAPTER 4

Character Recognition

This chapter describes work preliminary to the development of an experimental HMM system for the recognition of legal amounts. Many of these results have been published previously (Kornai, Mohiuddin and Connell [1995]). Our main goal of this chapter is to present techniques of feature set development and performance results for these various feature sets. Sections 4.1 - 4.7 discuss preliminary work involving the performance of Hidden Markov Models on the task of character recognition. Several properties of HMMs were established on the basis of this task and were used as guidelines for the work presented in the rest of this thesis. Section 4.7 then discusses the preparation of check images that will be used in the recognition experiments that will be presented in Chapter 5.

4.1 Introduction

There are a number of recognition techniques that have been shown to produce very good results on discrete characters. In order to assess the effectiveness of HMMs using different features, we used two of these techniques as baselines: Learning Vector Quantization (Kohonen et al. [1992]) and a multilayer perceptron system developed by Mohiuddin and Mao [1994]. The experiments presented in this chapter are on

discrete characters. Since much of the difficulty in cursive word recognition lies in the problem of segmenting a word into characters, this is an inadequate test of the HMM system's ability to recognize real world data. However, these experiments provide a starting point in which general properties of the HMM can be explored.

In the early stages of this research, two data sets were used to evaluate different feature sets. The first of these was part of a database released on CD-ROM from the Electrotechnics Laboratory of Japan (ETL). This data consists of 64 by 63 pixel single character images, over a thousand samples for each of 26 uppercase characters. The second set of data was taken from the NIST CD-ROMs. This data is made up of groups of 2100 single character bi-level images, one group for each of the lowercase characters *e f g h i l n o r s t u v w x y*. These characters were chosen because they are the ones that are contained in the common numeric value words found in the legal amount.

Approximately 10% of the ETL images contained characters that were labeled incorrectly or couldn't be distinguished by a human and were therefore discarded. Due to this, the ETL set contains much 'cleaner' data, while the NIST set should be considered more realistic. For the ETL experiments, sets of 500 and 1000 ETL images were used for training with the remaining 100 set aside for testing, while for the NIST experiments, sets of 1000 and 2000 images were used for training and 100 for testing.

4.2 Preprocessing

In their original form, the ETL images are not tightly cut to the characters contained within them, leaving a border of whitespace around the characters of varying width. To reduce these inconsistencies, the boundaries of the image were cut to the bounding box of the character. For each edge of the image, an algorithm starts at the edge and

examines each row (column in the case of left or right edges) moving inward. Once a row (column) containing black pixels is found, this is taken as the new boundary of the image. Images are next width normalized to 16 pixels. For this purpose, both a bilinear resampling technique and a bicubic resampling technique were tested. The results of each of these techniques was judged to be equivalent for this task.

Since we wish to work with bi-level images, the 8-bit greyscale ETL images are thresholded by an empirically determined value that removes most of the speckle noise, while leaving the character as undisturbed as possible. The effects of thresholding either before or after resampling were tested with no noticeable difference between the two.

To produce the final preprocessed form of both the NIST and ETL data, each character image was reduced to what has been named a *char* image. These are 24×16 binary images that are produced by applying a non-linear height subsampling to the character image that compresses pixels toward the top and bottom of the image into a smaller number of pixels at the top and bottom of the resulting image. The purpose of this non-linearity is to reduce the number of pixels containing the ascender and descender strokes, and emphasize the central region strokes.

4.3 Features

Four different categories of feature sets were used in comparing the effectiveness of HMMs to LVQ and MLP. This section describes these features.

4.3.1 50 Dimensional Features

The first set of features tested was a simple representation of the *char* images described in the previous section. The first two features encode the height and width of the character in the image, and the remaining 48 features are the values created by

grouping each set of 8 consecutive pixels in the *char* image into a byte. Since two numbers that are close in value may have binary representations that are drastically different, pixel-level similarities are not well represented. Therefore, the Gaussian estimators used in the HMM may not be able to model these features well.

4.3.2 Contour Direction Features

Another feature set derived from the *char* images are 72 contour direction features taken from the first 88 features of the Takahashi feature set (Takahashi [1991]). Later experiments at IBM have shown that 16 of these features, corresponding to the four corners of the image, can be omitted without any loss in the effectiveness on the features on discrete characters centered within the image. This is again demonstrated in Section 4.4 using LVQ. We will refer to these as the *CDF/88* and the *CDF/72* feature sets.

4.3.3 CO/12 and CO/24

Two feature sets were constructed by averaging the blackness within 12 or 24 horizontal strips within a sliding window. This created a 12-dimensional feature vector, known as the CO/12 features, and a 24-dimensional feature vector, known as the CO/24 features. The CO/24 features used in this chapter were computed simply using the non-linear subsampling technique described in Section 4.2, and with a window size set to a single pixel width and a step size of one pixel. This resulted in 16 feature vectors per character image. The CO/12 features were created from these CO/24 features by averaging together sets of two consecutive pixels. Other window and step sizes will be presented in Chapter 5.

4.3.4 LO10

Another feature set was created using a one pixel wide, one pixel step sliding window. These features track the vertical displacement of black lines. For each of the 16 columns in an image, the first five vertical runs in that column are stored as two values each: the center of gravity and the width of that run. If there are less than five runs found in a column, the remainder of the feature vector is filled with zeros. At ten features per column, this makes a total of 160 features per character image. These will be referred to as the *10*16* features.

Another way to represent these features is to make four 40-dimensional vectors per character image by grouping together four adjacent columns of the 10*16 features into each vector. These will be referred to as the *40*4* features.

4.3.5 Principal Component Analysis

Many of the features presented here may contain redundancies. Chapter 2 described how *Principal Component Analysis* (Hotelling [1933]) can allow us to project these features onto the space spanned by the eigenvectors of their covariance matrix, so that these redundancies can be eliminated. Additionally, the features in the new space that correspond to the largest eigenvalues will contain most of the original feature information, allowing us to reduce the overall dimensionality by eliminating features with little information. PCA for all experiments was accomplished using a component of the IBM semi-continuous HMM system (see Section 4.6).

4.4 LVQ

Learning Vector Quantization is a gradient descent vector quantization package produced by the Helsinki University of Technology. There are three learning algorithms

available with this package. The first of these, LVQ1, adjusts the nearest cluster center to an input vector, using some learning rate, toward that input value if both cluster and input belong to the same class, and away from that input value if they belong to different classes. An additional optimization to LVQ1, automatically adjusts the learning rate individually for each cluster based on its previous value and whether the current input is classified correctly. This is called OLVQ1. The second algorithm, LVQ2.1, makes similar adjustments, for each input, to the centers of two clusters at once, one of which must belong to the same class as the input and the other of which must belong to a different class. An additional restriction is that the input must fall within a defined distance window around the cluster centers. The third algorithm, LVQ3, adds to LVQ2.1 a stabilizing factor that, for each input, moves all cluster centers of the same class toward that input.

After experimenting with using different combinations of these algorithms in sequence, it was determined that, for this data, OLVQ1 obtains a high classification accuracy on the test set by itself, and use of the other algorithms beyond this do not improve this accuracy much. Results shown in Table 4.1 were obtained by using LVQ's interface program, lvq_run, using default parameters.

4.5 Multilayer Perceptrons

A multilayer perceptron system (MLP), presently in use by IBM's OCR group for the recognition of segmented characters was trained and tested on each of the two data sets in turn. These results (courtesy of Jianchang Mao) are shown in Table 4.1 with the LVQ results.

Table 4.1. A Comparison of Baseline Techniques vs. Single State HMM

Algorithm	Set/Size	Dim (orig)	% correct
LVQ	ETL/500	50	78.04
LVQ	ETL/500	72	65.48
LVQ	ETL/500	88	65.96
LVQ	ETL/1000	50	79.23
LVQ	ETL/1000	72	66.50
LVQ	ETL/1000	88	66.73
MLP	ETL/2000	88	99.01
MLP	NIST/1000	88	92.50
MLP	NIST/2000	88	93.81
HMM	ETL/1000	27 (72)	96.12
HMM	ETL/1000	23 (88)	96.96
HMM	ETL/1000	27 (88)	96.92
HMM	NIST/1000	27 (72)	88.00
HMM	NIST/2000	27 (72)	88.00
HMM	NIST/1000	23 (88)	85.56
HMM	NIST/2000	23 (88)	87.25
HMM	NIST/1000	27 (88)	88.25
HMM	NIST/2000	27 (88)	89.06

4.6 Hidden Markov Model

One of the HMM systems used here is part of an IBM project used for on-line handwriting recognition (Bellegarda et al. [1993]; Nathan et al. [1993]; Bellegarda et al. [1994]) that has been adapted to accept more generic feature vectors as input. This is a semi-continuous density HMM which uses unsupervised clustering to define discrete components of the data population. The distribution of these components are then estimated as a Gaussian distribution. The second system is a continuous density HMM which will be described in detail in Section 5.1.

Table 4.1 shows the results of the NIST and ETL data sets trained using the CDF features on a single state HMM using a single Gaussian mixture. Feature vectors are projected to a lower dimensionality using PCA, and their original dimensionality appears in parenthesis. An observation at this point is that the performance of the HMM is already better than LVQ and is approaching that of the MLP. However, the MLP is still significantly better than the HMM, obtaining as much as a 43% reduction in the error rate for the NIST data.

4.6.1 Multiple Mixtures vs. Full Covariances

The results presented in Table 4.1 are hardly a fair comparison since a single mixture density with a diagonal covariance is a weak model, unable to represent any interactions between the feature dimensions. The MLP and full covariance HMMs are capable of representing much more complex distributions than this. Table 4.2 shows the results of training a single state HMM on the NIST data using different numbers of Gaussian mixtures to deal with the complexity of the population distribution.

The best result now obtained by the HMM approach (27-dimensional CDF features using 40 mixtures) is 91.93%, an error rate improvement of 26% over the single mixture equivalent.

Table 4.2. The Impact of More Components in the Mixture

No. of Features	Dim	Train Set	10 Mix	20 Mix	30 Mix	40 Mix
10*6	27	1000	**79.19**	79.69	80.25	80.38
10*6	27	2000	**80.44**	81.88	82.94	82.12
10*6	30	1000	77.56	**80.12**	80.75	81.69
10*6	30	2000	78.62	**82.50**	82.44	82.31
10*6	32	1000	**78.00**	79.81	79.00	80.50
10*6	32	2000	**79.94**	82.19	82.81	82.44
10*6	35	1000	78.56	79.44	78.88	79.88
10*6	35	2000	79.31	82.31	82.25	83.25
40*4	40	1000	76.94	79.19	79.56	79.06
40*4	40	2000	79.19	80.44	80.00	80.88
CDF/88	27	2000	90.12	**91.31**	91.62	91.93

Another method of creating more powerful models is by using full covariance Gaussian mixtures. Section 3.6.2 discussed some of the issues involved in the use of full covariances models versus mixtures of diagonal covariance models. Table 4.3 shows the results of training on the NIST data using single state, single mixture, full covariance model HMMs (these are marked with an **F**). As can be seen by comparison with Table 4.2, the best results obtained are still diagonal covariance models. The point at which increasing the number of mixtures has beaten the results obtained by full covariance models are shown in boldface. Feature sets of dimensionality greater than 32 could not even train a single full covariance mixture. It is apparent from these results that the flexibility of the diagonal covariance models can be of great advantage over the full covariance models in some cases.

4.6.2 The Effects of Multiple States

Up until this point we have been looking at single state HMMs. While this provides a good comparison of the power of an HMM to that of other discrete character techniques, this is not what the HMM is meant to be used for and does not use it to

Table 4.3. The Impact of PCA

dim*columns	after PCA	states	1000 examples	2000 examples
10*16	27	1	69.44	70.75
10*16	27	1F	78.88	78.62
10*16	30	1	n.a.	68.31
10*16	30	1F	77.94	77.94
10*16	32	1	68.81	70.75
10*16	32	1F	77.56	77.56
10*16	35	1	68.75	70.31
40*4	10*4	4	68.31	69.31
40*4	10*4	7	73.94	75.62
40*4	15*4	4	68.56	69.31
40*4	15*4	7	75.56	76.06
40*4	20*4	4	66.81	66.88
40*4	20*4	7	71.75	72.31

its fullest ability.

Table 4.3 shows the results of training models with a range of states for training sets of size 1000 and 2000. The LO10 features are used to demonstrate the effects of dividing up the character between states. The 10*16 vectors are used in a single state HMM by concatenating all 16 columns together into a single 160-dimensional feature vector. As can be seen, the reduction in dimensionality due to PCA decreases as the LO10 vectors are spread among states. However, an increase in recognition accuracy is obtained as the number of states increases. The best single mixture results presented in this table (76.06% correct) are the 7 state, 15-dimensional (reduced from 40) feature vectors. A much more effective set of features for multi-state models were the CO/12 and CO/24 features. These are shown in Table 4.4 for 8, 12, and 16-state models.

Although these results do not live up to the performance of the CDF features, the CDF features tend to rely on the proper segmentation of characters. The CO/12 features on the other hand do not rely on segmentation, but take a loss in performance

Table 4.4. Recognition Rates for Multi-State HMM

set/size	feat/dim	states	% correct
ETL/1000	CO/12	8	90.12
ETL/1000	CO/12	12	93.35
ETL/1000	CO/12	16	95.08
ETL/1000	CO/24	8	88.73
ETL/1000	CO/24	12	92.24
ETL/1000	CO/24	16	94.19
NIST/1000	CO/12	12	73.38
NIST/2000	CO/12	12	76.06
NIST/1000	CO/12	16	79.12
NIST/2000	CO/12	16	79.81

in return.

Since increasing states and increasing mixtures are both methods of increasing the total number of parameters in the model, the effects of each were explored. Increasing the number of parameters places a greater burden on the training data, so we should attempt to gain as much performance as possible per parameter.

Assuming a standard left-to-right model with self transitions, next state transitions, and single state jumps, the number of state transition probabilities that must be estimated by the model are $2k - 3$, where k is the number of states. The number of parameters required by each diagonal variance mixture is $2kd$, where d is the dimensionality of the features. Table 4.5 compares the effects of increasing the number of states and the number of mixtures on the NIST/1000 set and the NIST/2000 set. Points where there is a rise in recognition accuracy in spite of a decreasing number of parameters are shown in boldface. These almost always occur when the number of states is increased (with one exception shown in italics).

Table 4.5. The Impact of k vs. m for Comparable Number of Parameters

Dim	Mixtures	States	Parameters	1000	2000
12	20	16	7709	84.38	85.00
12	20	12	5781	82.38	84.56
16	5	24	3885	**85.81**	**87.69**
12	10	16	3869	85.00	85.81
16	4	24	3117	84.25	**87.38**
12	10	12	2901	82.31	84.19
16	5	16	2589	**82.81**	**84.25**
16	3	24	2349	**85.62**	**86.44**
16	4	16	2077	82.88	84.62
12	5	16	1949	*84.62*	*85.69*
16	2	24	1581	83.69	**86.62**
16	3	16	1565	82.50	84.88

4.6.3 Tied Mixtures

The IBM semi-continuous HMM allows for the use of unsupervised clustering to create Gaussian mixture prototypes that can be used across many states and many models as tied mixtures. Since data clusters created by unsupervised clustering can draw from a data pool larger than that available to clusters created by supervised clustering, this should mean that better Gaussian prototypes can be created if the distributions that have these between-character and between-state similarities are truly close enough to each other.

A number of experiments were conducted to compare the effects of using clusters that were created separately for data from separate characters, but where state ownership of the data was ignored (character-level supervised clustering), with the effects of using clusters created separately for data from separate characters and separate states within the character model (state-level supervised clustering).

Figure 4.1 shows the results of an experiment using 10*16 LO10 features reduced to 27-dimensions. The plot of the total number of clusters used to create mixture

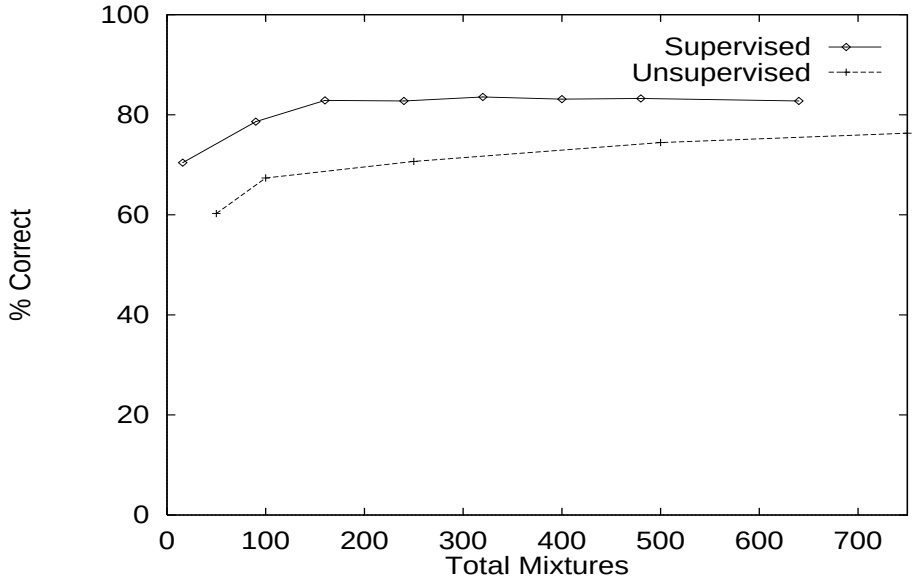

Figure 4.1. Supervised vs. Unsupervised Clustering

Table 4.6. State Level Supervised Clustering

states	char supervised	char & state supervised
1	53.50	55.06
3	55.62	66.12
5	61.75	73.00
7	64.75	75.62
9	65.50	76.62

models versus recognition rates shows two curves. The lower curve shows the result of using unsupervised clustering to create the mixtures, and the upper curve is based on character-level supervised clustering. This clearly shows the advantages of character-level supervised clustering in this domain.

To investigate the effects of state-level supervised clustering, experiments for supervised and unsupervised clustering were conducted in which a single cluster was created for each state of each character using only the data from that state. The

results of this combined character and state supervised clustering can be seen in Table 4.6 in comparison to just character level supervised clustering. As can be seen, supervised clustering at the state level is also advantageous for this type of data.

4.7 Check Data

This section contains a description of the collection, preprocessing, and truthing (labeling) procedures for the check images that will be used in training a legal amount recognition system. Recognition experiments performed on this data will be presented in Chapter 5, but this information is provided here in order to show the difference between the ideal case, working with presegmented characters, and the realistic case, working with images in which segmentation is difficult.

4.7.1 Data Collection

One of the deficiencies in much of the work done to date on legal amount recognition is the lack of research involving real-world data. Good results have been reported on legal amount data that has been collected by asking volunteers to fill out fake checks (Guillevic [1995]). This type of data may suffer from the following shortcomings:

1. Often the number of writers for which data is collected is limited to a very small number of people.

2. Often only one type of check format is used, while many check formats exist in the U.S. These are all very similar for the most part, but a variety of check formats should be considered so their dissimilarities are not overlooked in the model created.

3. People may tend to write more or less neatly than on real checks knowing that these will be used for a study.

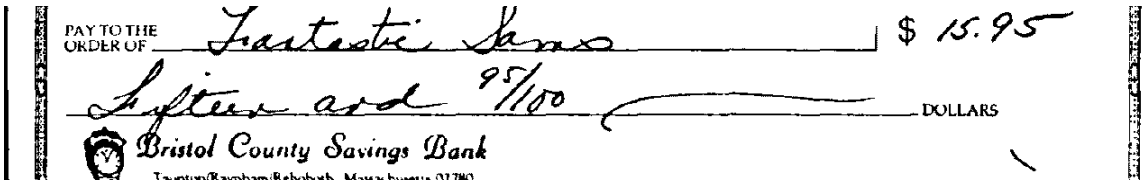

Figure 4.2. An Example Check from the Database (confidential data removed)

4. The prior probability distributions estimated using the checks will not necessarily reflect the population of real-world checks.

5. Time restrictions usually limit the amount of data that can be collected.

If these factors are overlooked, the data collected will not be representative of the population of real world checks. To avoid these shortcomings, it would be desirable to work on a large database of real checks selected from the population of real-world checks at random.

Over 20,000 scanned check images were obtained for use in ongoing research at IBM. These images were selected at random from a pool of US checks processed by a commercial system which recognizes the numeric, or *courtesy amount*, field. This data includes a large number of writers and a variety of check styles. In addition, the quality of these scanned images is characteristic of that which would be found in a production recognition system since the scanner of such a system was used in creating these images.

The style of writing from one check to another check varies greatly among these images. Some are written using discrete, but possibly touching characters, some are written in purely cursive writing, while most seem to be written using some combination of discrete and cursive. Additionally, the grammatical style in which a writer writes an amount may or may not include such things as the word *dollars*, the word *and*, a hand drawn horizontal line at the end and/or beginning of the amount,

an ampersand, or hyphens. The amount in cents may be written as a fraction or as the amount followed by the word *cents*. Common variations in the way the fraction is written include a denominator written as '100' or 'xx', and the amount of tilt in the fraction from '00/100' to '$\frac{00}{100}$'. Furthermore, different bank's checks vary in the height of the legal amount line, and the size, location, and overall design of the logo.

4.7.2 Image Thresholding

In one experiment the effects of two different thresholding methods were tested: the adaptive thresholding method developed by Niblack [1986] and a global thresholding method. The global thresholding method works by computing an intensity histogram for a greyscale image and searching for the primary and secondary intensity peaks that correspond to the background and the writing respectively. The primary peak is found as the grey level in the histogram that has the largest value. The secondary peak is found by searching from the end of the histogram corresponding to the darkest grey level, and searching inward until a local maximum in the histogram is found. The histogram is smoothed using a lowpass filter so that a false peak is not picked up by this search. For each peak, the standard deviation (σ) of the curve underneath the peak is calculated using the histogram values on the outer slopes of the peak (blacker side for the black peak and whiter side of the white peak). The global threshold is then set as the point that lies an equal number of standard deviations from the two peaks:

$$x\sigma_{primary} = x\sigma_{secondary}$$

To test the effectiveness of this method, 256 level greyscale images of the single digit numbers $0-9$ were taken from the ETL database. Preprocessing of these images consisted of first binarizing the image, then subsampling the image to normalize the

Table 4.7. Comparing Two Different Thresholding Methods

Thresholding Method	Recognition Accuracy
Global-Global	0.65
Global-Niblack	0.565278
Niblack-Global	0.787037
Niblack-Niblack	0.863889

height, after which another step of binarization is required. These two different stages of binarization give four possible combinations of thresholding to compare: global-global, global-Niblack, Niblack-global, and Niblack-Niblack. For the purposes of comparing these four different combinations, a set of models was trained for each using 1200 examples of each digit. Table 4.7 gives the recognition results of testing on an independent set of 216 examples per digit. It is apparent from this example that Niblack's method is the best candidate for both stages of binarization.

4.7.3 Finding the Legal Amount Zone

The format of US checks is neither so rigid as found in most forms processing applications, nor so loose as to require page layout analysis techniques. The zone finding method used here (Kornai and Connell [1996]) is a combination of top-down and bottom-up approaches which relies on statistical information about the structure of the images to restrict the search space. This information takes the form of parameters which were empirically determined by a gradient descent search, but are not necessarily optimal.

The legal zone is found in reference to the *pay_to* and *legal amount* baselines in the image. In a first attempt to find these baselines, horizontal black pixel runs are computed over a range of rows, and clustered into peaks. The legal amount and pay_to lines tend to appear as strong peaks, with the legal amount line most often the

strongest of these peaks, and rarely found outside of the top three peaks. Parameter values restrict the possible pair of peaks that can be identified as the legal amount and pay_to lines based on their absolute location as well as their distance to each other. Other restrictions imposed by the parameters include run lengths and densities, as well as peak sizes and densities.

Due to minor errors in the scanning process, a check may be skewed within its image. If the amount of skew is significant (defined here as $> 1\%$), it should be corrected. The amount of skew can be detected based on horizontal black pixel runs. Beginning points as well as end points are compared for consecutive runs, and the slope between corresponding points is computed. The image skew is then taken as the average of these slopes. Rather than use this calculated skew to actively correct the image, the skew is used *passively* by taking it into consideration in subsequent processing steps.

Sometimes the baselines can not be determined based on runs as previously described, in which case a second method is attempted. Once the skew has been found, projections can be computed in the direction of the skew rather than purely horizontal. Peaks are then extracted and processed as in the run based method described above.

In order to eliminate the pay-to line, the legal amount zone is defined to start just below this line. This zone then extends down to a distance below the legal amount line that is equal to one half the distance between the legal and pay-to lines. This is done in order to catch any descenders. After preprocessing, the images contain only this zone, now with the legal amount line located two thirds of the way down the image.

4.7.4 Slant Detection

Legal amount writing, as most cursive writing, tends to have a significant amount of slant. This slant differs from writing to writing and therefore should be normalized. In order to reduce computation, slant detection is performed on a reduced image, one eighth the width and height of the original, created by dividing the image into 8-by-8 pixel blocks and summing. Scanning left to right across the image, projections are calculated in the direction of 21 different slants from -45 degrees to 45 degrees. The top three projections for any slant are added and the slant with the largest count is taken as the slant value.

In most experiments, the horizontal range in which this slant was searched was limited to an empirically determined range in order to avoid including non-written objects that may dominate the slant calculation. An example of another method to pick this range will be presented in Section 4.7.8.

4.7.5 Characteristics of Data

There are many points which could prove to be troublesome in attempting to recognize the legal amount in these checks. The first of these is scanning/binarization noise. The images may contain speckle noise, or broken/faded lines due to inefficiencies in the scanning processes and thresholding of the image. Another difficultly is the intrusion of descenders from the pay-to line into the legal amount zone. These may even be overlapping words that we are trying to recognize, making them appear as ascenders for those words and thus further adding to the confusion. Similarly, the legal amount descender zone (the lower $\frac{1}{3}$ of the preprocessed image) may contain noise such as bank logos, or writing from the *memo* or the *signature* lines. The legal amount line itself, although helpful in finding this zone, becomes another source of noise when trying to recognize the words since the point where the line intersects the

Table 4.8. The Difference Between Two Humans on Word Segmentation

Tolerance (pixels)	4	8	12	16	20	24	28	32	48	64
% Within	51.1	83.0	89.4	89.4	91.5	91.5	93.6	93.6	97.9	97.9

words will not always be exactly the same. There also exists vertically running black pixel lines at the left and right of the image due to a combination of two things: a frame type of design which may exist around the border of the check, and edge noise where the scanned image extends beyond the edge of the actual check.

Attempting to perform recognition of the images at the character level could be very difficult. Many of the characters are barely visible in the word, appearing as a short, indecipherable wave, or perhaps as a high velocity scribble when combined with neighboring characters. Not only would this make recognition difficult, but to have a human discern reliable segmentation points for use as ground truths would be very hard. In fact, it was discovered during the truthing process that even deciding upon proper segmentation points for the words was difficult at times due to horizontal word overlap. This is primarily due to the slant of the writing causing the lower left of one word to be started below the upper right of the previous word. In one experiment, the truths produced by two different humans on the same data were compared. Table 4.8 shows the correspondence rate between the two for a range of acceptable error tolerances. Each column in this table can be read as the percentage of examples that two humans agree upon within x pixels. As Table 4.8 shows, even two humans do not agree over 90% of the time to within 20 pixels ($\frac{1}{12}$ of an inch), and not over 95% of the time until the error tolerance is above 32 pixels ($\frac{1}{8}$ of an inch).

4.7.6 Truthing the Images

It is in this preprocessed format that the images were truthed. The *truths* for these images consist of a horizontal starting point within the image, an ending point, and label for each item in the image. These items may be handwritten items, such as words, fractions, or horizontal lines, or they may be items that are part of the check form itself, such as black edge lines, the machine printed word 'Dollars', or whitespace. A full list of category labels, their descriptions, and the number of occurrences of that category in the first 4030 images can be seen in Table 4.9.

In order to assist the process of truthing, a tool was created by modifying a popular shareware image viewing program called `XV`. With this tool an image is displayed and delimiters are marked, moving from left to right, by clicking with the mouse. These define the points of segmentation between different items. After marking a segmentation point, the label of the proceeding item is typed at the keyboard. The coordinates of the items and their respective labels are then stored in a file for later processing. For speed purposes, the tool accepts a list of images to be loaded and processed one at a time without having to exit the program.

4.7.7 Bootstrapping

Initially, a large portion of the time spent by a human in producing these truth labels was spent on categories such as *whitespace*, *edge noise*, *machine printed 'Dollars'*, and not on the important categories such as the numerical value words. Once a large number of these common categories have been labeled, an HMM can be trained to recognize and isolate what we have called the *content zone*. This is the area in which numerical word are contained and it is the job of this HMM to recognize the left and right edges of this zone and extract it - this is one of our performance criterion which will be discussed further in Section 5.3.1. For this purpose, an HMM using the

Table 4.9. Original Word Categories from Truths

Label	Description	Instances Available
ws	Large whitespace	19839
ms	Between-word whitespace	5414
bl	Left edge noise	4955
ds	Machine printed *Dollars*	3422
fr	Fraction	3037
br	Right edge noise	2244
ln	Handwritten horizontal line	2854
and	*and*	1654
100	*Hundred*	866
dhs	*Dollars*	803
1	*One*	730
-	Hyphen	698
20	*Twenty*	691
aps	Ampersand	664
5	*Five*	532
30	*Thirty*	503
50	*Fifty*	442
2	*Two*	355
40	*Forty*	340
lnds	ln overlapping ds	293
6	*Six*	282
3	*Three*	272
4	*Four*	266
9	*Nine*	255
7	*Seven*	252
8	*Eight*	241
60	*Sixty*	216
70	*Seventy*	212
10	*Ten*	175
80	*Eighty*	151
frln	fr overlapping ln	131
wave	Very wavy line	131
90	*Ninety*	115
15	*Fifteen*	102
cents	*cents*	89
lnfrds	ln overlapping fr overlapping ds	67
13	*Thirteen*	57
14	*Fourteen*	56
11	*Eleven*	55

Table 4.9. (cont'd).

Label	Description	Instances Available
18	*Eighteen*	55
1000	*Thousand*	52
12	*Twelve*	52
no	*no*	48
17	*Seventeen*	47
19	*Nineteen*	47
16	*Sixteen*	42
lnfr	ln overlapping fr	41
only	*only*	26
frds	fr overlapping ds	18
0	*Zero*	4
even	*even*	3
$	Dollar Sign	2
andln	and overlapping ln	2
frlnds	fr overlapping ln overlapping ds	2
andfr	and overlapping fr	1
centsds	cents overlapping ds	1
none	*none*	1

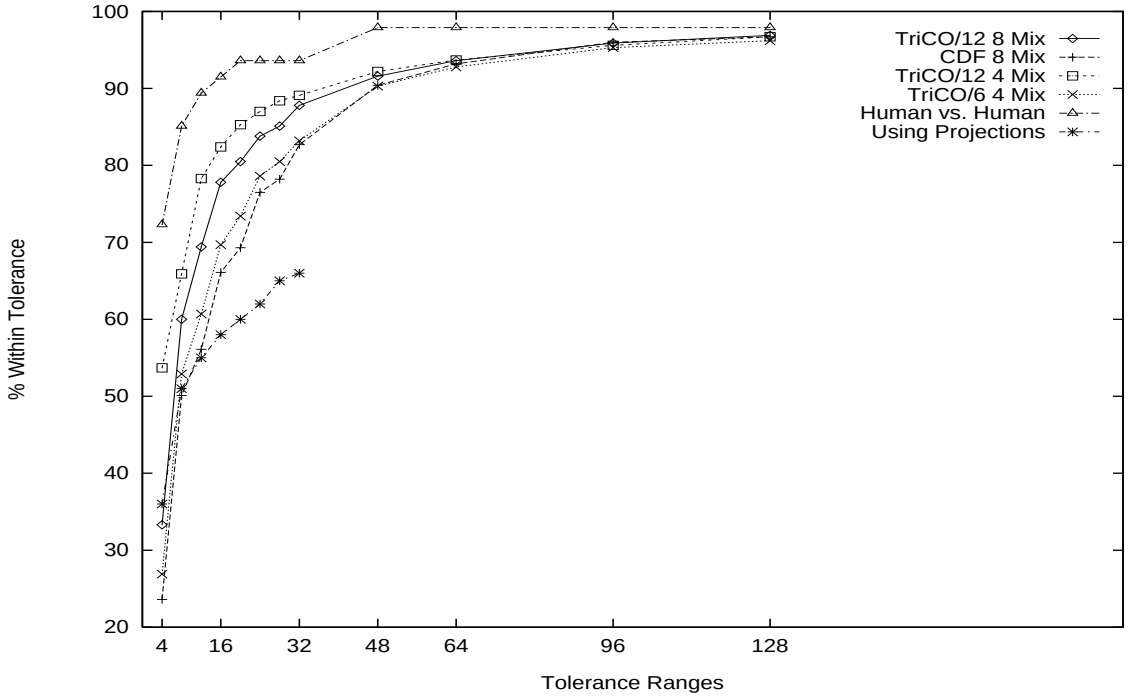

Figure 4.3. Content Zone Segmentation: Left Side

8 mixture *TriCO/12* features reduced to 18 dimensions by PCA (to be described in detail in Section 5.5.3) was used. Using this model, both the left and right bounds of the content zone can be found within 0.533 inches (128 pixels) of the truthed values for 85.6% of the images.

Reducing the image to these extracted zone images previous to truthing reduced truthing times to approximately $\frac{1}{3}$ the time required for the entire zone. This includes time saved by loading the smaller images of the extracted zones.

Figures 4.3, 4.4, and 4.5 present the accuracy of four different HMMs used as content zone segmenters in finding the left of the zone, the right of the zone, and both left and right at once (each of these HMMs will be described in Chapter 5). As a comparison, we show the results of segmentation using a simple vertical projection technique which searches for segmentation points where the magnitude of the projection crosses some threshold. Also, shown again for comparison is the amount of

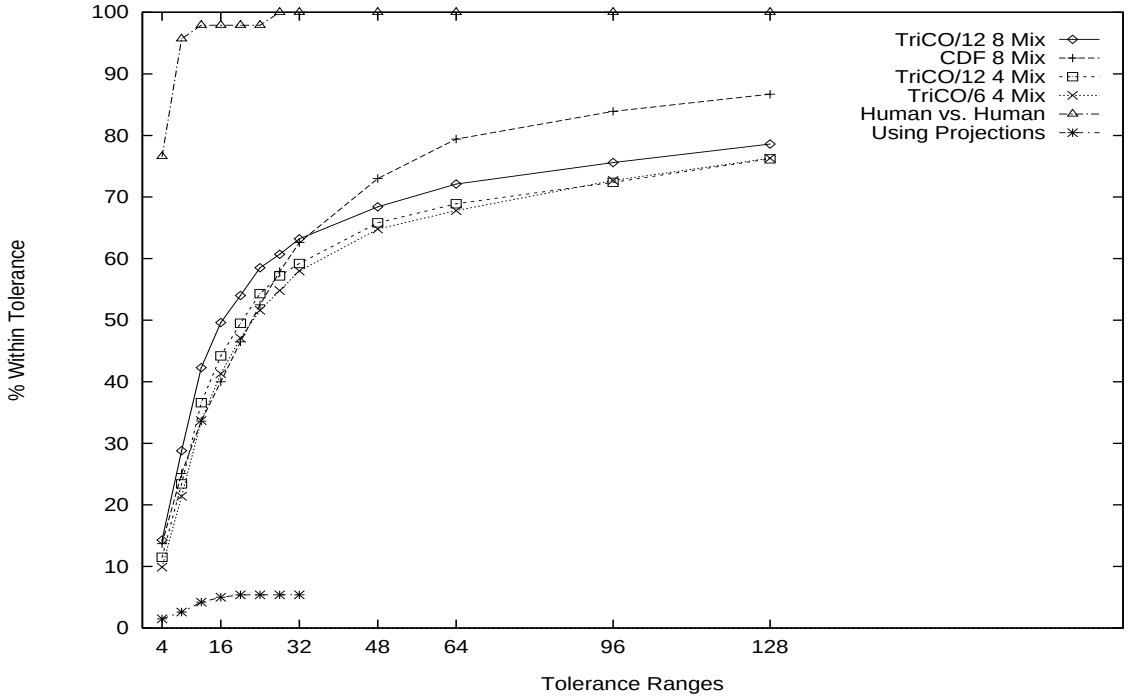

Figure 4.4. Content Zone Segmentation: Right Side

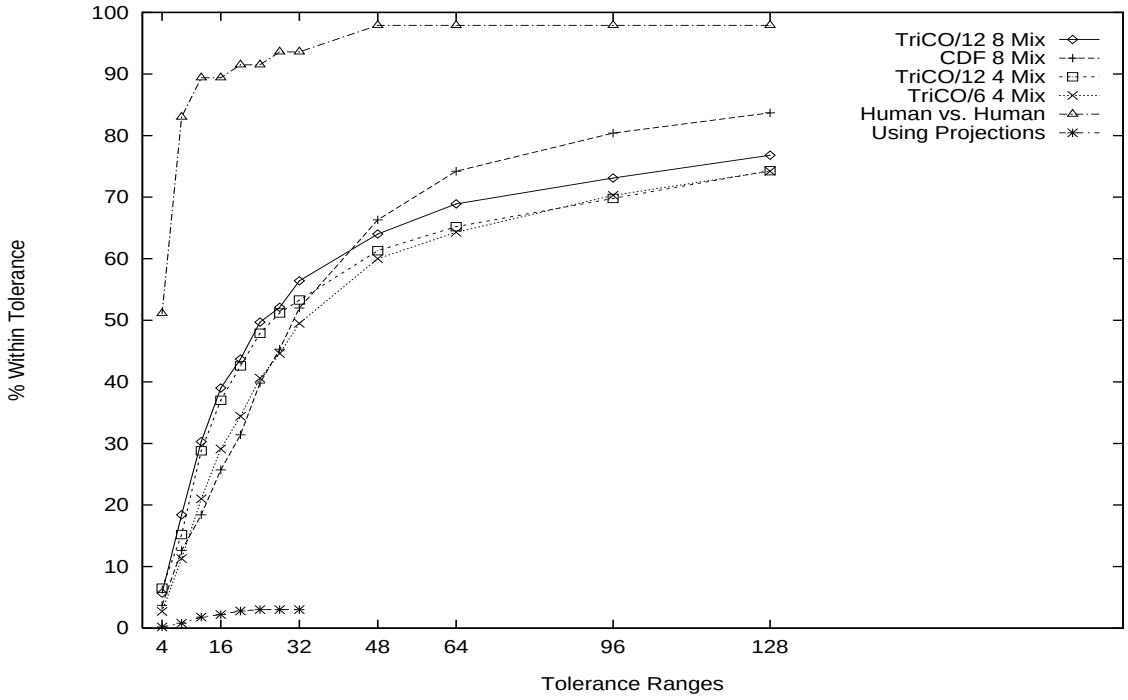

Figure 4.5. Content Zone Segmentation: Both Sides

accuracy found when comparing two humans on labeling the same images. As can be seen, the HMM technique performs much better than the projection technique.

Not surprisingly, comparisons of Figures 4.3 and 4.4 show that both the HMM and the projection methods are much better at finding the left side of the content zone than the right side. This is no doubt due to the fact that in most cases the content is preceeded by whitespace, while it is most often followed by a handwritten horizontal line, more handwritten words, or the machine printed word *Dollars*. What is surprising is that the humans in our study perform better on the right side than on the left side. One possible explanation for this is that the whitespace on the left side causes the humans to be more sloppy when picking a segmentation point, while on the right side they must more carefully pick a segmentation point between two objects that are very close together.

4.7.8 Two-Pass Slant Detection

Slant detection, when attempted on the entire legal amount field, often produces undesirable results. Noise, such as edge noise which runs vertically, and descenders from the pay_to line, can contaminate the average slant calculation of the writing. Even closer to the center of the legal amount field, a handwritten horizontal line can contaminate the calculation. In order to avoid this contamination to as high an extent as possible, only the area in which handwritten words appear should be used for slant detection. Calculation of the slant over an empirically determined horizontal range (see Section 4.7.4) may prove to be unreliable since it is difficult to define a range that will exclude such objects as handwritten horizontal lines. By using the extracted content zones, the slant can now be calculated using only that area of the legal field in which it has been determined that handwritten words appear. This requires extra processing time, since two passes of recognition must now be done: one to extract the zone in which the slant should be calculated, and a second to obtain the final results,

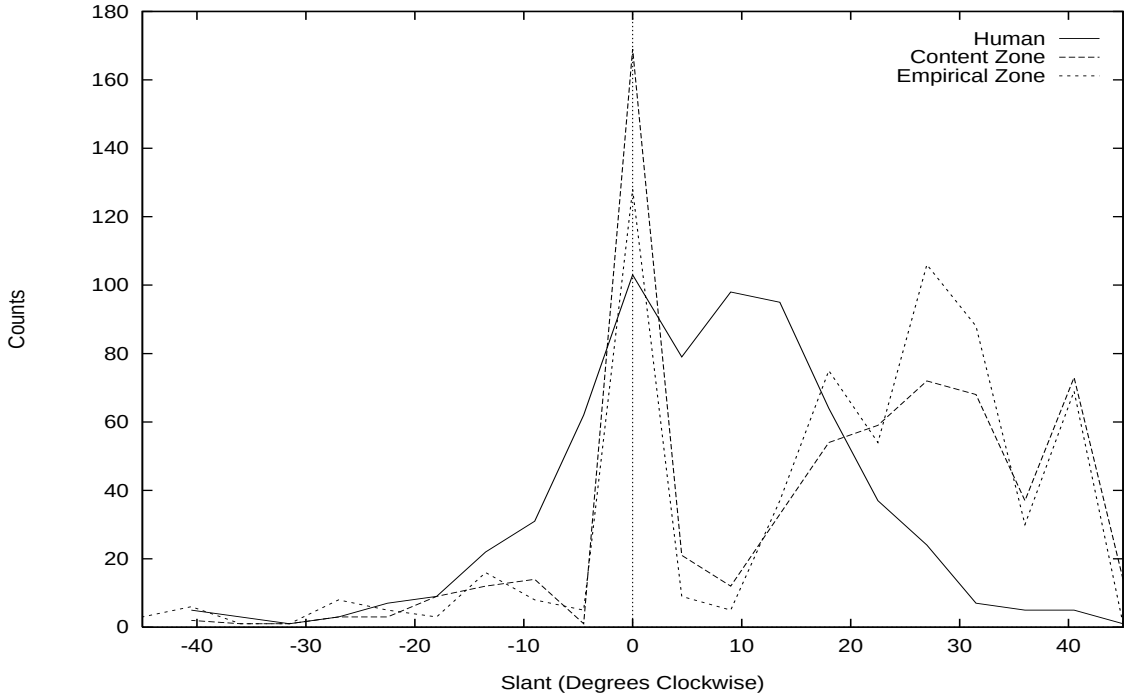

Figure 4.6. Distribution of Slants Measured

using the slant corrected image. This extra stage can only be justified if these stages can be pipelined, and if the accuracy of the slant detection is improved in a way that significantly benefits the accuracy of the system.

The performances of the two different slant detection techniques were compared to the performance of a human using a protractor to measure the dominant slant of an image displayed on a computer screen. Figure 4.6 shows the distribution of slants measured by each of these methods on 658 test images. From this figure, it can be seen that the two automated methods have secondary peaks at a much greater angle than that of the secondary peak for the human. Figure 4.7 shows the angular difference between slant estimates by a human and each of the two slant detection methods. It can be seen in these two figures that the content zone method of slant correction is in a higher degree of agreement with the human than the empirical zone method.

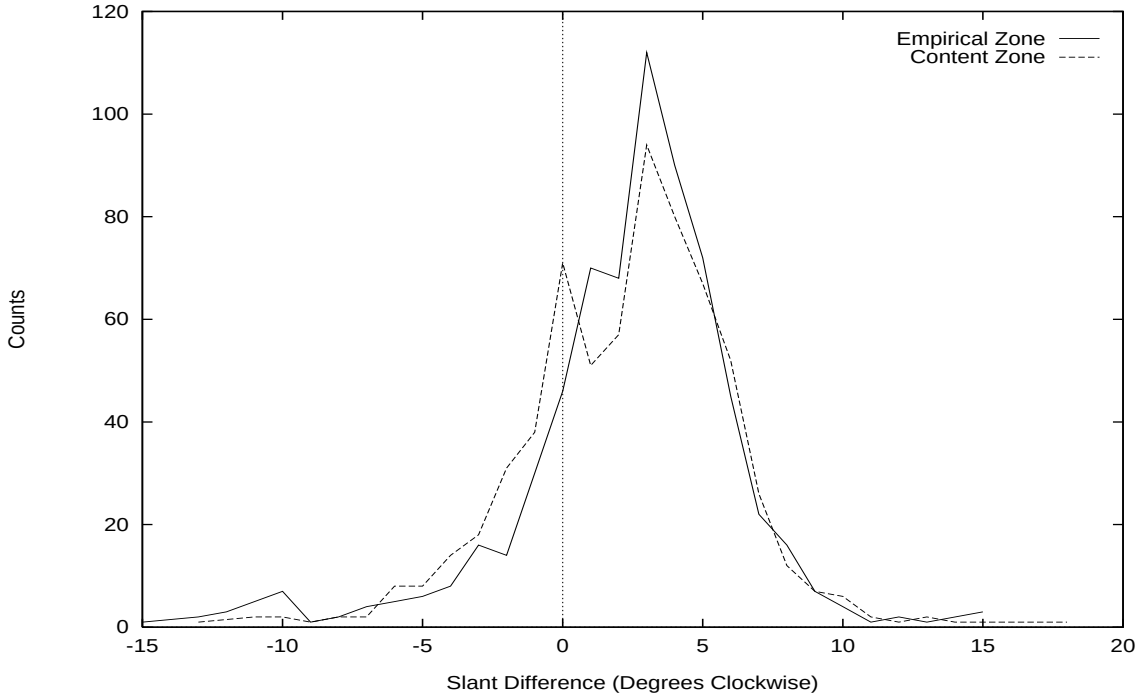

Figure 4.7. Slant Differences from Human

After viewing many slant corrected images, it has become apparent that a large portion of the difference between the human and these automated slant detection methods comes from a conservativeness on the part of the human in selecting an angle. This may be in part due to a combination of the difficulty in measuring an angle that is displayed on a screen, and the degree to which humans are more comfortable viewing handwriting which is slightly slanted. The most prominent errors on the part of the automated methods can be attributed to diagonally running fraction lines which tend to dominate the slant calculation, but these cases were only shown to cause more than a slight deviation in slant approximately 2% of the time. Other than this, a subjective quality comparison of the three methods has shown no significantly higher quality of slant selection from the human than from either of the automated methods, with the *Content Zone* method performing slightly better than the other two.

4.8 Summary

In this chapter, a number of feature sets have been introduced and experiments have been conducted on presegmented character data. An evaluation of three recognition techniques, Learning Vector Quantization, a multilayer perceptron, and a single state Hidden Markov Model has shown that the HMM is already capable of producing better recognition results than LVQ, and comes close to the performance level of the MLP. Other experiments have established a number of properties about HMMs in this domain. In particular, an increase in the number of diagonal covariance mixtures can provide an increase in recognition accuracy, sometimes greater than that provided by the use of a full covariance matrix. Given the data available, only a certain number of parameters can be reliably trained and these parameters must be used in a way such that the benefit to the model is maximized. It has been shown in this chapter that an increase in the number of states can provide a greater benefit to the system than increasing the number of Gaussian mixtures. In order to more effectively utilize parameters, the possibilities of parameter tying, in the form of tied mixtures, was explored. However, a comparison of creating Gaussian mixture estimates based on groups of data obtained from different levels of clustering supervision has shown that supervised clustering was better both at the character and state level for this type of data.

A description of the check data collection and preparation has been given, including a description of our zone finding, skew detection, and slant detection algorithms. In addition, global thresholding was compared to Niblack's method of adaptive thresholding, with Niblack's method producing the best results. Finally, the results of word segmentation using HMMs were compared to that of a simple projection method and two humans. It was determined that the HMM performed much better on this task than the projection method, and the difficulty of the task was seen in the amount of

discrepancy between the two humans.

CHAPTER 5

Word-Level Recognition

The work described in the previous chapter, based on character recognition, acts only as an indicator of the types of features that can be expected to work well on cursive words. There are two ways of modeling words. One uses individual character models, tied together in a network defined by some grammar that allows for the creation of all words in the lexicon. The second method is to create a separate model for each word in the lexicon. The first method has the advantage of requiring only a small number of different models to be created (one for each character) that can be used to model a large lexicon. The disadvantage of this method is that coarticulation effects are lost. This can be remedied by creating n-gram models for sequences of n characters that have large coarticulation effects. The extreme of this is the second method in which n-grams are created where the value of n is not constant, it is the length of the word. If the lexicon is small, the second method becomes feasible and can provide more accurate models of the individual words. However, this can require much more data to train since the character examples from different words in the lexicon cannot be shared in a straightforward manner.

5.1 The Recognizer

The recognizer used for these experiments is a commercial HMM toolkit sold by Entropic Research Laboratory, Inc., called HTK. Although this toolkit was primarily created for work with speech, it was easily adapted to work with our sliding window features for cursive handwriting. HTK includes facilities for model initialization using a Viterbi aligner, Baum-Welch training of models, and a Viterbi recognizer which accepts a grammar, for simultaneous recognition and segmentation of a chain of words. The grammar acts as a language model, tying models together to form phrases. See Young, Woodland and Byrne [1993] for more details on the implementation of these algorithms.

Model prototypes must first be created that set the number of states, state connectivity, number of mixtures, and the feature dimensionality. In the multi-channel case, these last two items are defined for each channel. It is these prototypes that are used by the initialization program to create an initial Viterbi alignment before beginning Baum-Welch training. After much experimentation, it was decided that this initialization stage was not necessary since equivalent results could be obtained without it.

5.2 Grammars

Two types of grammars were implemented for testing. The first of these, referred to as the *Forced* grammar, is different for each image and is simply a concatenation of the labels as they were truthed for each respective image. This is used to test the segmentation abilities of these models with much of the added complexity of classification eliminated.

The second grammar type is not so static as the first, defining production rules from which almost any check amount can be constructed. There are three such

grammars that have been used in the research presented here, and they will each be described in the remainder of this section. The first two of these focus on measuring the segmentation ability of the models, while the third concentrates on the task of legal amount recognition. All three of these grammars were created empirically based on viewing the ground truths of a number of images. Using this type of grammar, a much greater demand is placed on the HMM to recognize the items in the image being tested.

5.2.1 N7 Grammar

To reduce processing time requirements while evaluating a wide range of features, the number of images used as training examples was limited to approximately 4000 for many of the experiments. In order to train good models, our target was to obtain at least 1000 examples for each category. As was seen in Table 4.9, there are few categories for which enough examples exist. To compensate for this, seven new categories were created, as shown in Table 5.1, which are combinations of the original categories.

Figure 5.2 shows the grammar that has been created to describe phrases that can occur using these categories. It should be noted that all words of interest, in terms of legal amount recognition, as well as words that commonly occur with them have been rolled into a single modeled called X.

5.2.2 N10 Grammar

The results of different models on the content zone segmentation task (to be described in Section 5.3.1) were compared based on three different criteria: the magnitude of error for the left edge of the content zone, the magnitude of error for the right edge, and the magnitude of error on the worst of the left and right edges. While many

Table 5.1. Combined Word Categories for the N7 Grammar

Category	Contents	Instances Available
ws	White Space	25254
X	Ampersand, *$*, *cents*, *only*, *no*, *even*, *and*, Handwritten *Dollars*, and all numbers	10758
bl	(Black) Edge Noise	7196
ds	Machine Printed Word *Dollars*	3423
fr	Fraction	3037
ln	Handwritten line (both *ln* and *wave*)	2985
hy	Hyphen	697

Table 5.2. N7 Grammar

$$LeftSide\ Center\ RightSide$$
$$Center \Rightarrow \mathbf{X}\ (\mathbf{X} \mid \mathbf{ws}\ \mathbf{X})^*$$
$$LeftSide \Rightarrow \mathbf{bl\ ws\ bl\ ws} \mid \mathbf{ws\ bl\ ws} \mid \mathbf{bl\ ws} \mid \mathbf{ws}$$
$$RightSide \Rightarrow PreDollar\ \mathbf{ds}\ PostDollar$$
$$PreDollar \Rightarrow WL \mid WL\ \mathbf{fr}\ WL \mid \mathbf{fr}\ WL$$
$$PostDollar \Rightarrow \mathbf{ws} \mid \mathbf{ws\ bl\ ws} \mid \mathbf{ws\ bl} \mid \mathbf{bl\ ws}$$
$$WL \Rightarrow \mathbf{ws} \mid \mathbf{ws\ ln} \mid \mathbf{ws\ ln\ ws} \mid \mathbf{ln} \mid \mathbf{ln\ ws}$$

of the models trained obtained reasonable results for the left edge, the right edge proved to be more difficult. This is because items to the left of the content zone are most often edge noise and whitespace, often with a long easily discernable whitespace just before the content zone. On the other hand, the writing to the immediate right of the content zone is often a fraction or a line, both of which should be harder to distinguish from an X word than a whitespace. Any whitespace between the right edge of the content zone and this next writing is usually very short, therefore making the right edge less obvious.

To assist the process of finding the right edge of the content zone, models were created for three words that are most frequently the last word in this zone, and therefore should define its right edge: *dhs*, *and*, and *aps*. When using the N7 grammar, these three words are included in the definition of category X. One problem with grouping a number of words from different categories into the definition of a single model is that if these categories are very different, then the amount of variability in the model increases. This decreases the strength of the model. As a simple example, if most of the examples used to train X are lengthy words (such as seventeen, hundred, etc...), then the model will prefer longer words over shorter ones and it will be very difficult for the model to recognize a word such as 'and'. Even if it finds the correct word, it will have difficulty finding the correct boundary locations for the word. Since finding the correct location of these words is crucial, it may be worthwhile to create separate models for them at this stage.

With the addition on these three new word models a new grammar is defined which has been found to cover the truths for all but approximately 2% of the images. This grammar can be seen in Table 5.3. It is described in terms of five major components: the main body of legal amount (*Body*), the portion of the legal amount containing edge noise (*Enoise*), the portion containing handwritten lines (*Lnoise*), a machine printed 'Dollars', and an optional fraction (square brackets denote an optional item). The

Table 5.3. N10 Grammar

$$\textit{Enoise}\ \textit{Lnoise}\ \textit{Body}\ \textit{Lnoise}\ [\mathbf{fr}]\ \textit{Lnoise}\ \mathbf{ds}\ \textit{Lnoise}\ \textit{Enoise}$$
$$\textit{Body} \Rightarrow \mathbf{X}\ (\mathbf{X}|\textit{Lnoise}\ \mathbf{X})^*\ \textit{Lnoise}\ [\mathbf{dhs}]\ \textit{Lnoise}\ [\mathbf{and}|\mathbf{aps}]\ \textit{Lnoise}\ (\mathbf{X}|\textit{Lnoise}\ \mathbf{X})^*$$
$$\textit{Enoise} \Rightarrow [\mathbf{ws}]\ (\mathbf{bl}\ \mathbf{ws})^*\ [\mathbf{bl}]$$
$$\textit{Lnoise} \Rightarrow ([\mathbf{ln}]\ (\mathbf{ws}\ \mathbf{ln})^*\ [\mathbf{ws}]) \mid ([\mathbf{hy}]\ (\mathbf{ws}\ \mathbf{hy})^*\ [\mathbf{ws}])$$

improvement provided by this grammar over the N7 grammar was rather significant. However, some experiments that will be presented in this chapter were conducted with the N7 grammar.

5.2.3 N28 Grammar

This grammar stands as the most complete in terms of the number of words individually modeled. A list of each category's contents and the number of examples used to train that category's model for different training set sizes used can be found in Table 5.4 and Figure 5.1, respectively. It should be noted that the 'teens' had to be grouped into a single model (called *Xteen*), as did the numbers 0, 11, 12, and 1000, and the words *cents*, *only*, *no*, and *even* (called *X12*). The grammar appears in Table 5.5. Since we wish to use this grammar to perform recognition of the legal amount, for which these combined categories are meaningless, these two categories have been left out of the grammar. The words 1 - 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 all are independent models with labels matching their number.

Table 5.4. 30 Category Word Models

Model Label	Contents
1, 2, 3, ...	*Numbers 1 - 9, and decades 10 - 100 respectively*
X12	1000, 11, 12, 0, cents, only, no, even
Xteen	13, 14, 15, 16, 17, 18, 19
dhs	*Handwritten word 'Dollars'*
ds	*Machine printed word 'Dollars'*
fr	*Fraction*
hy	*Hyphen*
ln	*Handwritten line*
and	Handwritten word 'and'
aps	*Ampersand*
bl	*Black Edge Noise*
ws	*White Space*

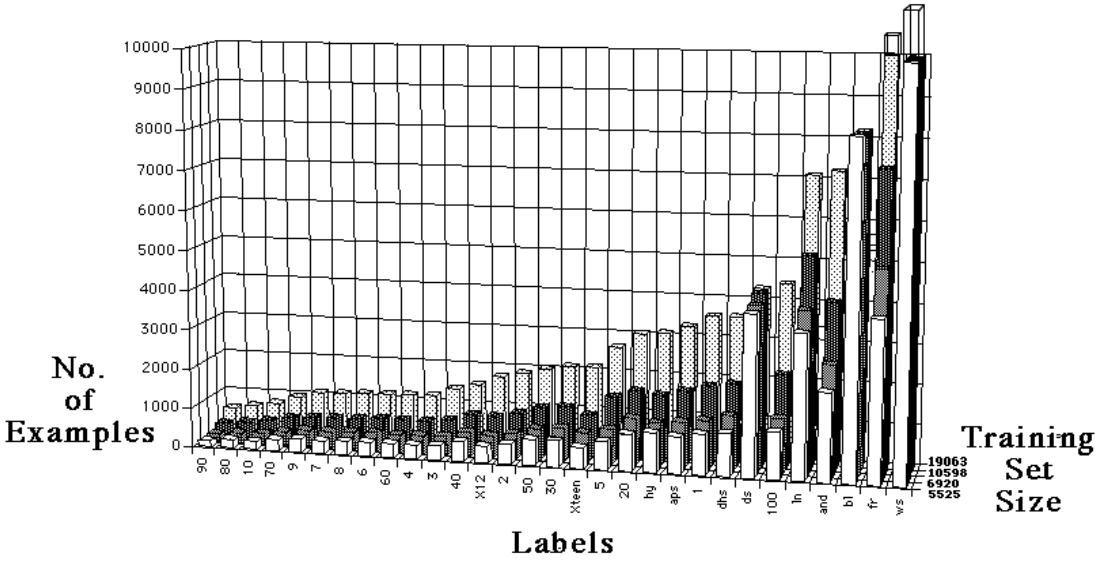

Figure 5.1. 30 Category Word Counts for Different Training Set Sizes

Table 5.5. N28 Grammar

$$\begin{gathered}
\textit{Enoise Lnoise Body Lnoise } [\mathbf{fr}] \textit{ Lnoise } \mathbf{ds} \textit{ Lnoise Enoise} \\
\textit{Body} \Rightarrow \textit{ Threedig} \mid \textit{Twodig} \mid \textit{Onedig} \\
\textit{Onedig} \Rightarrow \textit{ Dig } [\mathbf{ws}]\ [\mathbf{dhs}]\ [\mathbf{ws}]\ [\mathbf{and} \mid \mathbf{aps}] \\
\textit{Twodig} \Rightarrow \textit{ Decade}[\mathbf{ws}]\ [\mathbf{hy}]\ [\mathbf{ws}]\ [\textit{Dig}]\ [\mathbf{ws}]\ [\mathbf{dhs}]\ [\mathbf{ws}]\ [\mathbf{and} \mid \mathbf{aps}] \\
\textit{Threedig} \Rightarrow [\textit{Dig}]\ [\mathbf{ws}]\ [\mathbf{hy}]\ [\mathbf{ws}]\ \mathbf{100}\ [\mathbf{ws}]\ [\mathbf{and}]\ [\mathbf{ws}]\ [\textit{Twodig}] \\
\textit{Dig} \Rightarrow \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9} \\
\textit{Decade} \Rightarrow \mathbf{10} \mid \mathbf{20} \mid \mathbf{30} \mid \mathbf{40} \mid \mathbf{50} \mid \mathbf{60} \mid \mathbf{70} \mid \mathbf{80} \mid \mathbf{90} \\
\textit{Enoise} \Rightarrow [\mathbf{ws}]\ (\mathbf{bl}\ \mathbf{ws})^{*}\ [\mathbf{bl}] \\
\textit{Lnoise} \Rightarrow ([\mathbf{ln}]\ (\mathbf{ws}\ \mathbf{ln})^{*}\ [\mathbf{ws}]) \mid ([\mathbf{hy}]\ (\mathbf{ws}\ \mathbf{hy})^{*}\ [\mathbf{ws}])
\end{gathered}$$

5.3 Testing

In order to test the effectiveness of the models, a testing utility included with HTK was used. This program accepts a grammar to specify how models should be interconnected end to end, creating a large network of states in which special *enter* and *exit* states are used to keep track of where one model ends and the next begins. A Viterbi alignment of the feature sequence is calculated for each test image. The sequence of models traversed in this alignment defines the items recognized in the image, while the distance into the input sequence in which a transition from the exit state of one model to the enter state of the subsequent model defines the segmentation point between these two items.

To ensure that our results are independent of the choice of test set, we first ran an experiment comparing two different test sets. Figure 5.2 shows N10 Grammar tests on two sets both of size 732. As can be seen, the results are only slightly different for the two sets. For further tests, we used the combined set of 1464 images.

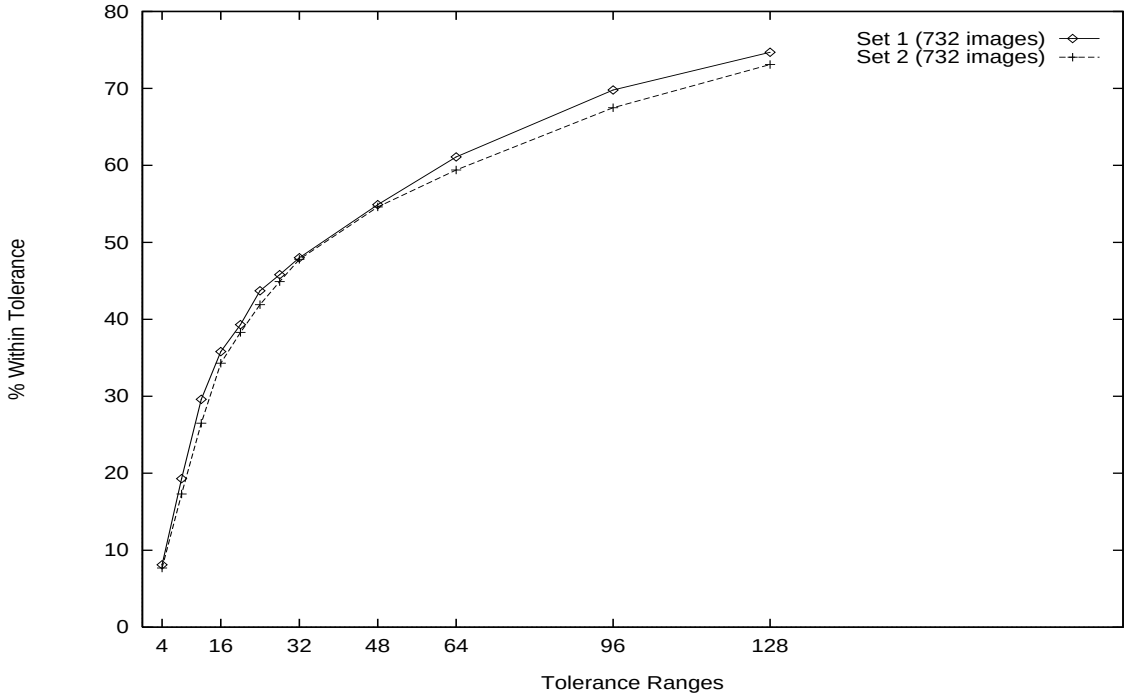

Figure 5.2. Testing on Two Different Sets

5.3.1 Performance Criteria

Two different criteria for measuring the performance of a set of models have been defined. The first of these focuses on the segmentation power of the model, while the second concentrates on the word-level and phrase-level classification accuracy.

Segmentation Measure

Since the most important items in the image are the words which form the dollar amount, a subset of the X category, a zone was defined starting at the left edge of the leftmost X items, and extending to the right edge of the rightmost X item. This zone which includes all X items in the image, we refer to as the *content zone*. We then want to test the ability of the model to distinguish the border between content words and other words or objects in the image, and measure this by how well the left and right edges of this zone are found. A scoring program provides a count for each side (left

and right) of the content zone indicating the error, in pixels, of the recognized edges versus the truthed edges. These counts are grouped into eight different categories based on whether they fall within 12 respective error tolerances: 4, 8, 12, 16, 20, 24, 28, 32, 48, 64, 96, or 128 pixels. At 240 dpi, these work out to be errors of 0.01667, 0.0333, 0.05, 0.0667, 0.0833, 0.1, 0.11667, 0.1333, 0.2, 0.2667, 0.4, and 0.5333 inches, respectively. A comparison of features sets using this performance measure should weigh more heavily their performance in a error tolerance range below 48 pixels.

Classification Measure

The classification performance was measured by a utility included in the HTK Toolkit which compares the output of the trained recognizer with the truths using a Dynamic Programming algorithm (Bellman [1957]; Bellman and Dreyfus [1962]). The *Word Correct* rate is calculated as the ratio of correctly recognized words, H, to the total number of words, N, defined by the truths.

$$\%WordCorrect = \frac{H}{N} * 100\%$$

This is calculated for H and N taken over all test images. This measure accounts for all errors that occur due to word substitutions or deletions. One category of error that is not represented are word insertions, that can occur when one word is broken into multiple words. Insertion rates are also listed in Table 5.6, and are calculated as the ratio of the number of insertions, I, to the total number of words.

$$\%Insertions = \frac{I}{N} * 100\%$$

The *Field Correct* rate is then the percentage of check images for which the entire legal amount (all numeric value words in the field) is recognized correctly. This is

limited to the dollar amount, because all fractions are lumped into a single model. On an average, there are approximately two content words per field. Therefore, if the rates of insertions and deletions in the field were both zero, the Field Correct rate should be approximately equal to $(P_{wc})^2$, where P_{wc} is the *Word Correct Rate*. Additionally, the Field Correct rate only counts the number of fully correct fields. This is not enough information in itself to compare two results since the number of partially correct fields is relevant when parameters are being tuned in a gradient descent type of optimization search. An effective comparison of these results must take into account the Word Correct rate, the Insertion Rate, and the Field Correct rate.

5.3.2 Test Data

For a given model and set of data for which it is trained on, the more closely representative the training data is (from the population of data we wish to model), the better a model can be trained for that population. Since we have no guarantee that the training data is fully representative of the population, tests should be conducted using test data from the population other than the data used in training. Testing on data from the training set can also be of some use, since this should show how well the model represents the data it has been trained on, and therefore could provide some indication of the model's capabilities if the data used to train it is identically distributed to the population from which it is taken. If two sets of models are trained on the same training data and then tested on that same data, their comparison still is valid. However, an important point here is that the results of testing on training data and testing on separate test data are not comparable.

A word should be said about the meaning of the recognition rates for test data. Given a model with a set topology, and a set number of parameters, and training set, one view of testing this model on its training data is that these results represent a

theoretical maximum accuracy achievable for that model. In order to better train the model, more examples (i.e., training data) are required. As more data is added to the training set, the results of testing on the training data should, if anything, get worse since increasing the size of the training set will cause it to more accurately represent the distribution of the population - which is most likely a more complex distribution then what can be reasonably modeled by a fixed size model. On the other hand, this does not represent the maximum accuracy achievable for the task of modeling the population of data, since the addition of more data to the training set may also allow a more complex model to be trained (e.g., one with more Gaussian mixtures, or full-covariance mixtures) which may better model the data.

The results presented in this chapter are tested using independent test sets for the most part. Those experiments for which such a test set was not available at the time, will specify whether they have been tested on training data.

5.4 Optimizing HMM Parameters

Although Baum-Welch training attempts to optimize some of the parameters of an HMM given the training data, other parameters must be set from the beginning. These kinds of parameters include the number of features, the number of states in the model, the number and weights of Gaussian mixtures, and the connectivity of the states within the model.

Additionally, the choice of using full covariances or diagonal covariances for the Gaussian mixtures can have a great impact on the effectiveness of the model. It should be pointed out that although the covariance of the overall training data has been reduced to a diagonal matrix by PCA, this most likely does not hold true for the individual clusters of data represented within each mixture of each state, which are subsets of the entire training data set with their own covariances. However, while this

indicates that a full covariance matrix would better model the data, the requirements on the amount of training data needed to train these additional parameters may be too great (an increase from n to $\frac{n(n+1)}{2}$ parameters for the variance where n is the dimensionality of the features), thus restricting us to a diagonal covariance matrix.

Factors which effect the number of parameters are the dimensionality of the features, the number of Gaussian mixtures, the number of states, and the number of connections between states. The number of parameters that can be trained, in any of these forms, will be restricted by the amount of training data available.

A variety of model topologies were tested for each word in a search for a near optimal fit. This search was directed by observing the length distributions of the words. One method that worked particularly well was to adjust the model length (number of states) and number of possible state jumps based on the mean, minimum, and maximum of the word length distribution. These topologies are kept constant in all model comparisons presented in this chapter.

A point that comes up in comparison between any two models is the number of trained parameters. Tweaking the number and distribution of trainable parameters in any model until its maximal configuration is found is a tedious gradient descent process. Repeating this process for every feature set evaluated for the sole purpose of comparison would be a waste of time. A more reasonable approach is to compare results for models trained using different features in which the number of parameters in each model is the same. In this way we have some indication of how well the feature set can utilize the parameters available to it.

5.5 Segmentation Experiments

A number of feature sets were evaluated, all based on the CO/xx features, where xx can be any divisor of the original 24 features. These are formed by averaging

together consecutive dimensions of the feature vectors forming an xx dimensional feature vector.

5.5.1 Window size

One set of experiments dealt with the effects of different window sizes. HMMs trained using window sizes of 4 pixels, 8 pixels, and 16 pixels were tested, with step sizes adjusted to match. Figure 5.3 shows a comparison of the results of some experiments using the *forced* grammar with the N7 set of models to test on training data. Comparison of the CO/6 features in this figure show the optimal window size to be 8 pixels, and the worst to be 16 pixels. Additionally, Figure 5.3 shows how the accuracy is dropped when the feature dimensionality is doubled by using CO/12 features instead of CO/6 features. In order to further test the effects of reducing dimensionality, models were trained using CO/3 features. As can be seen by the example shown, using a window size of 16, this added compaction caused a drop in accuracy.

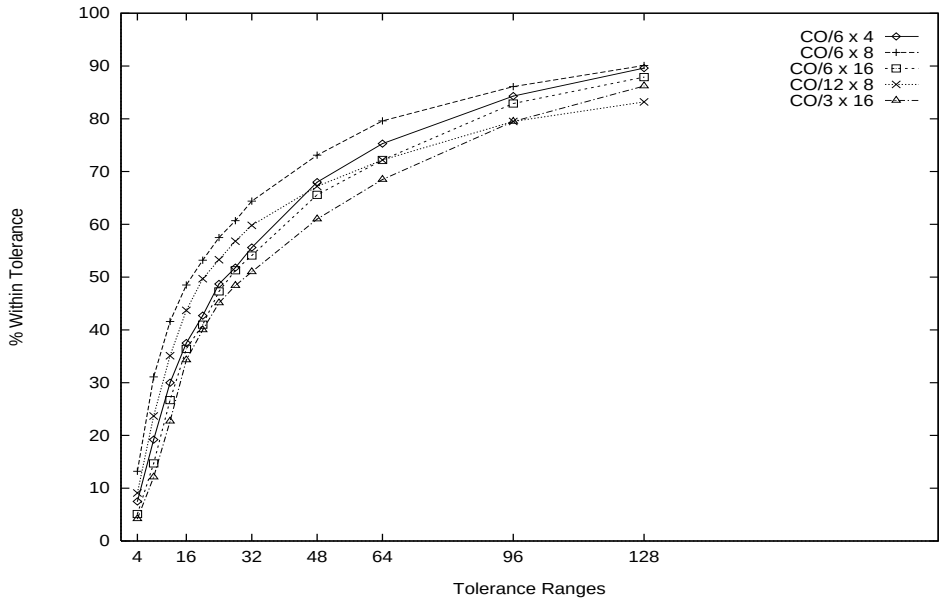

Figure 5.3. The Effects of Using Different Sliding Windows

5.5.2 Lowpass Filtering and Derivatives

In further attempts to improve these feature vectors, it was recognized that information on how observation vectors change as we move from left to right across an image are not captured by these features. Once the model is in some state, there are many ways that it may have gotten there, and the present observation may not give any clues as to what this previous context was. In this way, there is much contextual information which is not being represented. This is an inherent deficiency in first-order HMMs of this type. In order to try to include this type of information, the value of every observation was adjusted so as to include information from the past and previous observations. In one method, derivatives were computed for adjacent feature vectors by calculating a weighted sum of the current vector, the previous vector, and the next vector using the following formula:

$$D_t = 0.5F_{t-1} - 1F_t + 0.5F_{t+1}$$

where F_t is the feature vector for column t, and D_t is the vector of derivatives computed for feature vector F_t. These derivatives can be used alone or can be added to the end of the current vectors, creating a feature vector twice as long.

In other experiments, a lowpass filter was applied to the signal by averaging over three sequential observations as follows:

$$L_t = 0.2F_{t-1} + 0.6F_t + 0.2F_{t+1}$$

The segmentation accuracies of these features, using CO/6 single mixture full-covariance models, are presented in Figure 5.4. The tests shown here were all conducted using the *forced* grammar. As can be seen, the performance of the derivatives by themselves does not match up to the performance of the raw CO/6 features.

Neither do the lowpass features, which are by far the worst results shown here. A combination of the CO/6 and its derivative features was constructed (CO/6 + Deriv) producing a 12-dimensional vector. In this case we see a performance improvement over the CO/6 features alone, however this is not a fair comparison, since the CO/6 + Deriv models contain a higher number of trainable parameters ($\frac{(12\times12)+12}{2} + 12 = 90$ per state, as opposed to $\frac{(6\times6)+6}{2} + 6 = 27$ per state). A fairer comparison can be made by increasing the number of CO/6 mixtures to 3 (= 81 parameters) or 4 (= 108 parameters). As can be seen in Figure 5.4, by 2 or 3 mixtures, the performance of the CO/6 features has already caught up to the CO/6 + Deriv features. Therefore, the derivative features do not appear to add much additional information for the feature dimensionality they require.

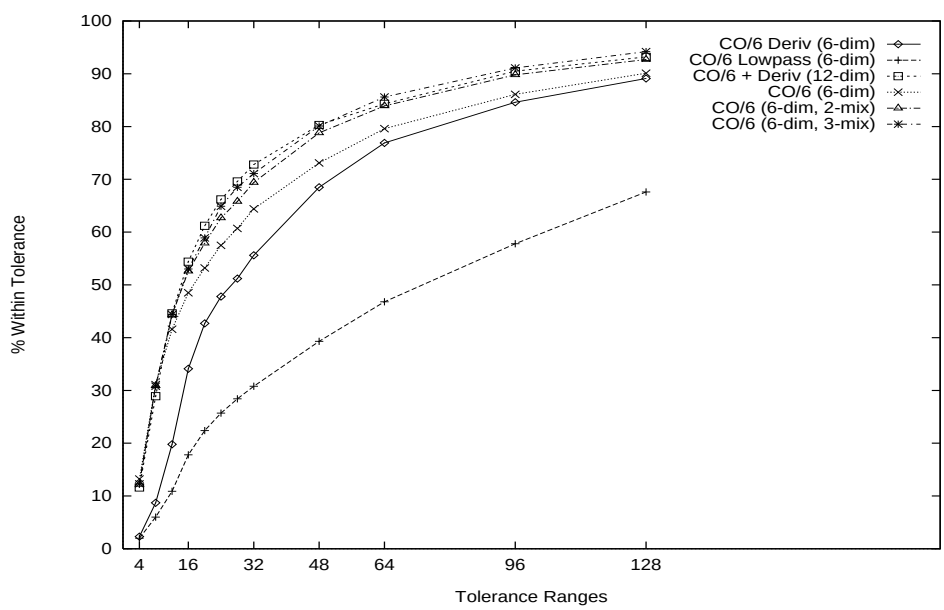

Figure 5.4. Derivatives and Lowpass Filtering

5.5.3 Triplication

The features presented in the previous section, based on derivatives and lowpass filtering, are both linear combinations of consecutive observations. As an alternative, the lengths of the feature vectors were tripled by creating each window's new feature vector as the concatenation of the previous window's vector, the present window's vector, and the next window's vector - a process we have called *triplication*. These new features (known as the TriCO/xx features, where xx is the dimensionality before triplication) then represent pixel trends across three columns (24 pixels) rather than trends in a single isolated column (8 pixels). This creates a very long feature vector - $xx \times 3$ features, many of which contain redundant information. Due to the 'curse of dimensionality', a reduction in the number of features is desirable while still retaining as much of the relevant information encoded in all the features. This is accomplished by a projection of the data to the principal components of its covariance matrix by *Principal Component Analysis* of the data (see Section 2.3.5). After PCA, we are able to greatly reduce the number of features while retaining most of the information. In the case of the CO/12 vectors triplicated (i.e., TriCO/12), this gives us a reduction from 36 dimensions to 18 dimensions while retaining approximately 95% of the original information. Using PCA on the triplicated vectors has the effect of including all the information from three consecutive vectors, and letting the covariance matrix show us what the most important components are, rather than us deciding as was done with the derivative features and the lowpass features.

Figure 5.5 presents a comparison of triplicated versus non-triplicated vectors on the CO/16 features. For the 4 mixture TriCO/16 features we have certainly surpassed the performance of the CO/16 features using 8 mixtures. This is a significant increase in performance without an increase in parameters (in fact, the number of parameters is slightly decreased). By switching to the TriCO/12 features, an even greater drop

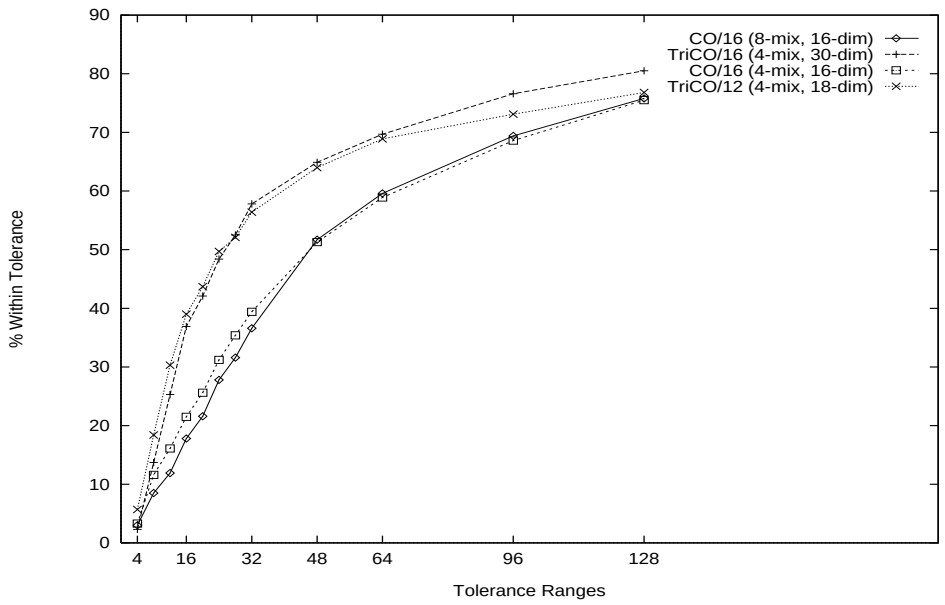

Figure 5.5. Triplication

in the number of parameters is achieved, while another significant increase in performance is realized for the smaller tolerance ranges. Here we also see our first case of a performance decrease when more mixtures are added in the 4-mixture CO/16 and 8 mixture CO/16 features. This is an indicator that at the 8-mixture level the model is being overfit to the data, and therefore a limit of performance for these features on a training set of this size has been reached. Figure 5.6 shows this limit for the TriCO/12 features, where our best results are on 8-mixture models.

5.5.4 N-tuplication

Having shown the effectiveness of the triplicated vectors, we attempted to expand this technique from three consecutive vectors to five vectors (*quintuplication*) and seven vectors (*septuplication*), thus including contextual information over a wider range. Figure 5.7 shows a comparison of triplicated, quintuplicated, and septuplicated vectors from one such experiment. It is apparent from the performance of the CO/12

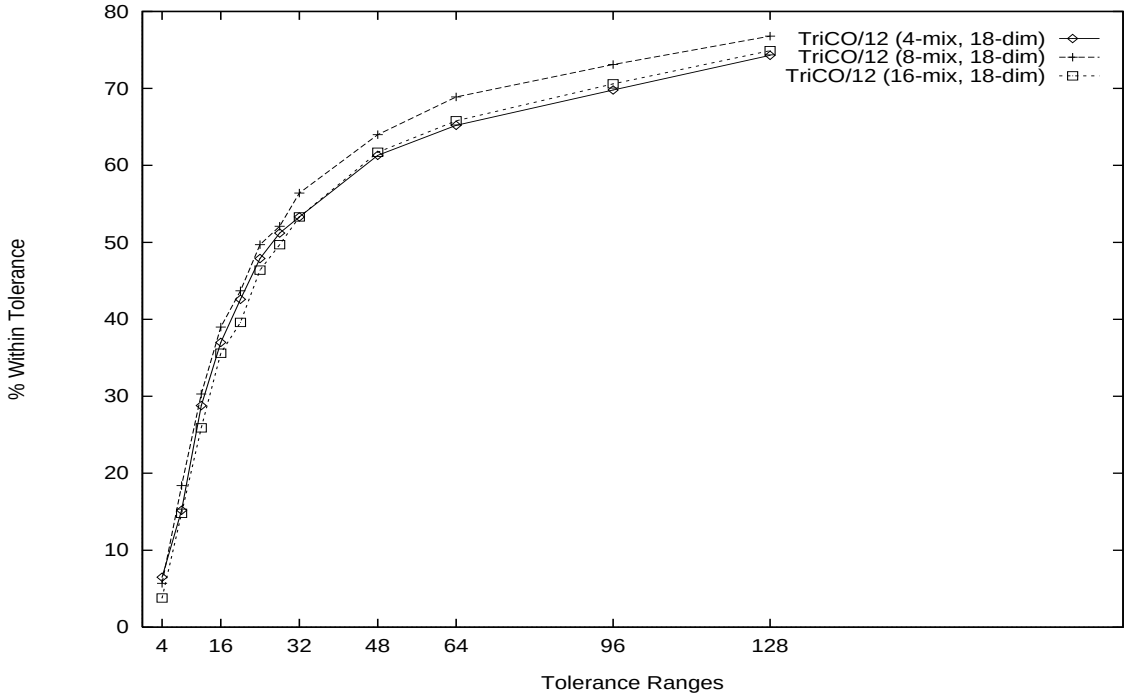

Figure 5.6. Triplication

features, that the benefits of using a wider range do not carry beyond triplication. Even the 2-mixture TriCO/12, which uses less than half the trained parameters of its quintuplicated and septuplicated equivalents, appears significantly better for the low tolerances.

5.5.5 Edge Features

In other experiments, two additional features were computed and added to the TriCO/12 and TriCO/6 feature vectors for a total of 38 or 20 features, respectively. These are the locations of the top and bottom of the blackness profile and are referred to as the *edge* features. The top is computed by starting at the upper-most row and searching downward counting the number of black pixels in that row until this number exceeds some threshold. This row number is then stored as the top. The blackness count is calculated from the present 8-pixel wide column plus a small percentage of

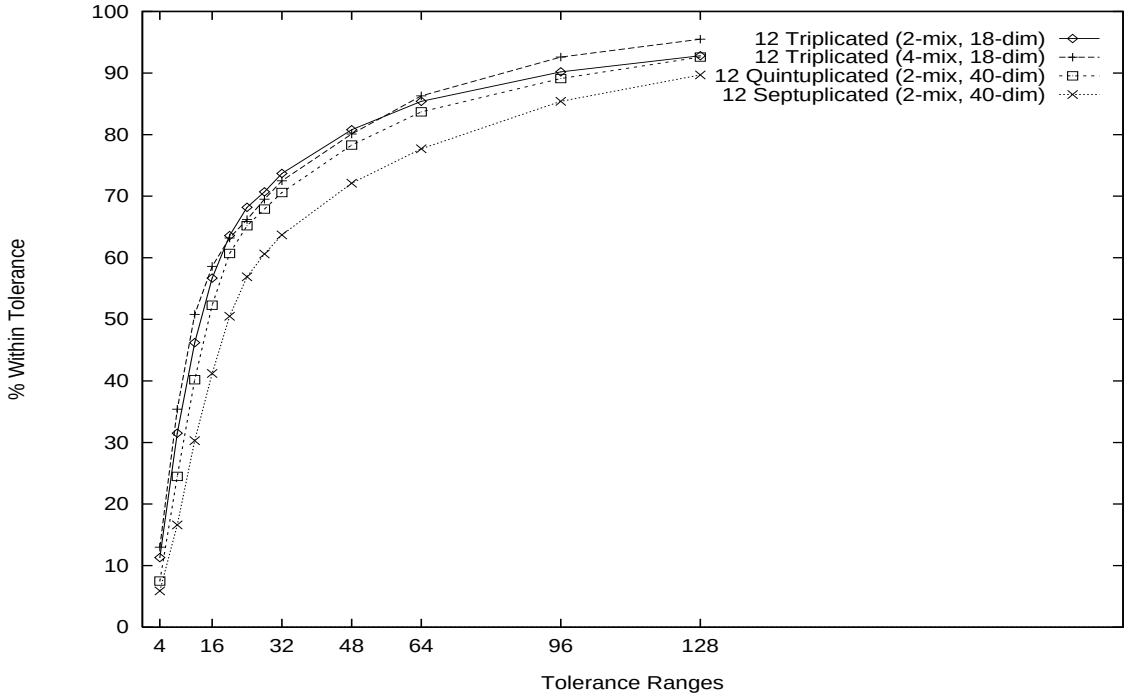

Figure 5.7. The Effects of Quintuplication and Septuplication

the column to its left and right so as to favor wide concentrations of blackness over thin ones. In addition, a large percentage of the blackness count in the previous row is carried over to the next row to prevent the algorithm from advancing too far into a faded word. The bottom location is computed in a similar fashion to the top location, starting at the bottom-most row and searching upward.

These new features were rotated down to 18 dimensions for the TriCO/12 features, and 12 dimensions for the TriCO/6 features. If the edge features provide any new information to the triplicated features, then this should be picked up by the PCA, thereby producing features that will perform better while retaining the same dimensionality. If they provide no useful information, then they should be pushed out by the PCA by being projected onto the weaker eigenvectors which will then be dropped in the dimensionality reduction.

The results of these experiments are shown in Figure 5.8. All features shown have

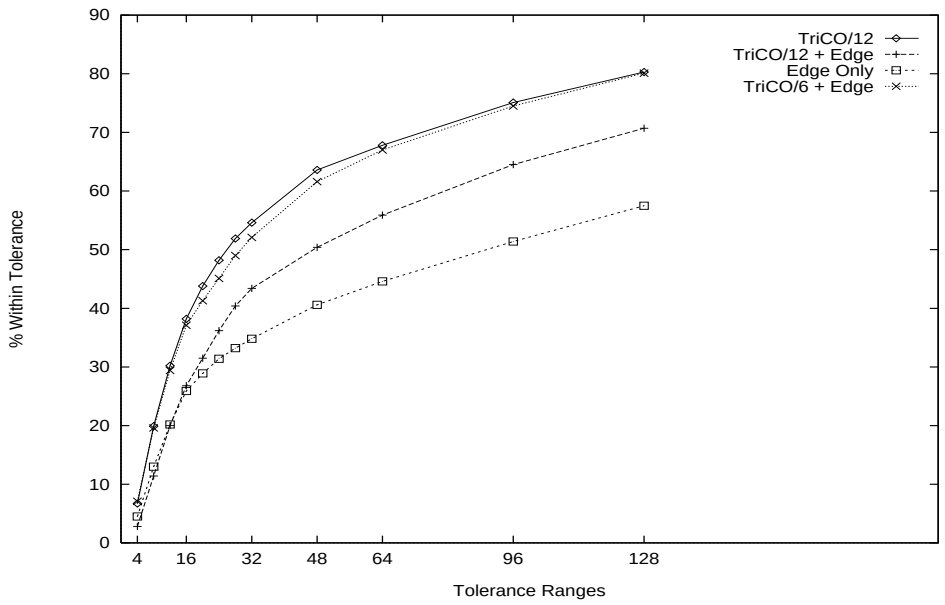

Figure 5.8. Edge Features

been triplicated. The *Edge Only* features demonstrate the power of these edge features by themselves. However, when added to the TriCO/12 features, their performance drops drastically. One possible reason for this is that most of the information contained in the edge features that is useful for discrimination may already be contained in the TriCO/12 features. Moreover, the information that PCA decides is important is not necessarily good for discrimination. If the edge features carry with them information that PCA views as important (i.e., a large variance), yet is unimportant for discrimination, when the dimensions of the combined features are reduced this useless information may become harmful by virtue of taking up space in the lower dimensionality that could have been occupied by other useful information that was present in the TriCO/12 features.

5.5.6 Fourier Features

Each vector of the CO/xx features can be thought of as a curve measuring the pixel depth over each slice (see Figure 5.9 which is a slightly modified version of Figure 1 by Elms [1994]). This curve can be broken down into periodic components using the Fourier transform. For these experiments, CO/16 features were transformed into a 32-dimensional vector in which each of the 16 Fourier coefficients was represented as two features: one for the real valued component, and one for the imaginary component. Figure 5.10 shows the results of one of these experiments. Using an equivalent number of parameters, a CO/16 model was tested containing twice as many mixtures as the 32-dimensional Fourier features (FFTCO/16). As can be seen, these features provide mostly equivalent results for small tolerances, and tend to fall below the CO/16 features as the tolerance increases.

Also shown in Figure 5.10 are the TriCO/16 features. A comparison of these with the FFTCO/16 features suggests that in order for the FFT features to be competitive with our best features, they must also include information from neighboring observations. Figure 5.11 contains the results of triplicating the FFTCO/16 features and reducing them down to 18 dimensions by PCA. A comparison with the TriCO/12 models of a similar number of parameters show that, although this triplication provides a great improvement in the Fourier features, they still do not provide an improvement over the much easier to compute TriCO/12 features.

In other experiments, the FFT of the TriCO/16 features was computed (called the FFT/TriCO/16 features) and reduced to 18 dimensions. The idea here is that any periodicity that exists between consecutive windows should be captured by these features. As an additional step, the effects of triplicating these features was tested, forming the TriFFT/TriCO/16 features, which were then reduced to 64 dimensions. These

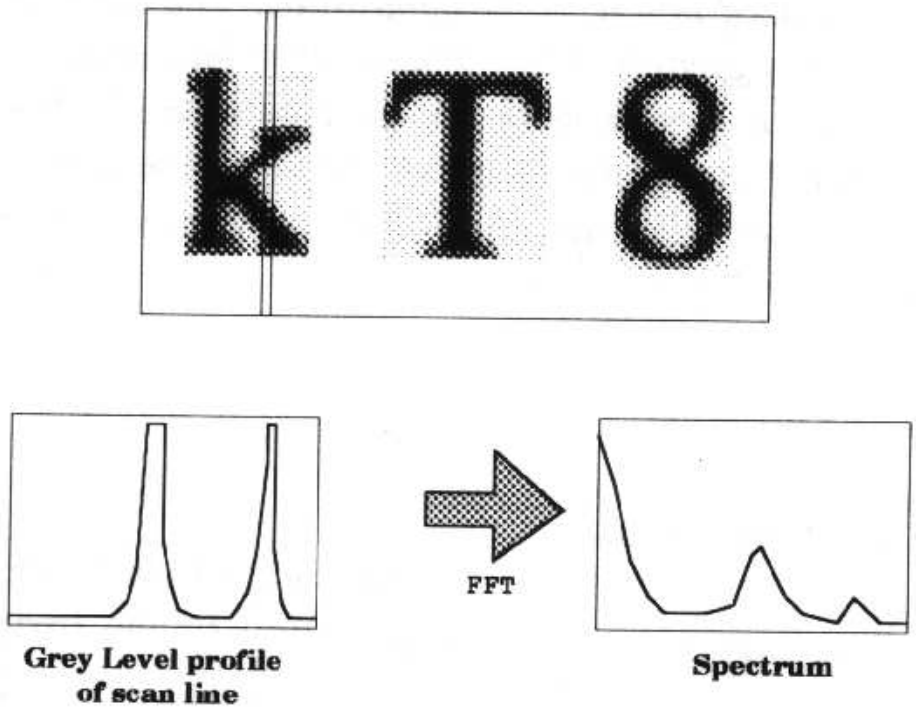

Figure 5.9. FFT Feature Calculation

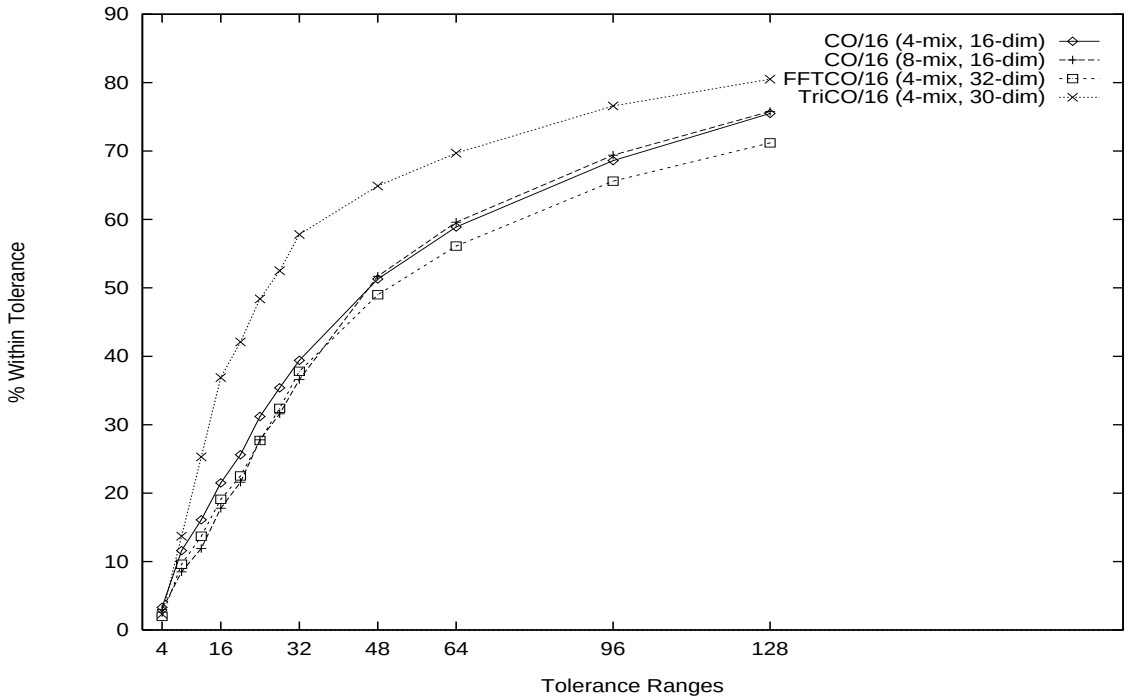

Figure 5.10. FFT of Features

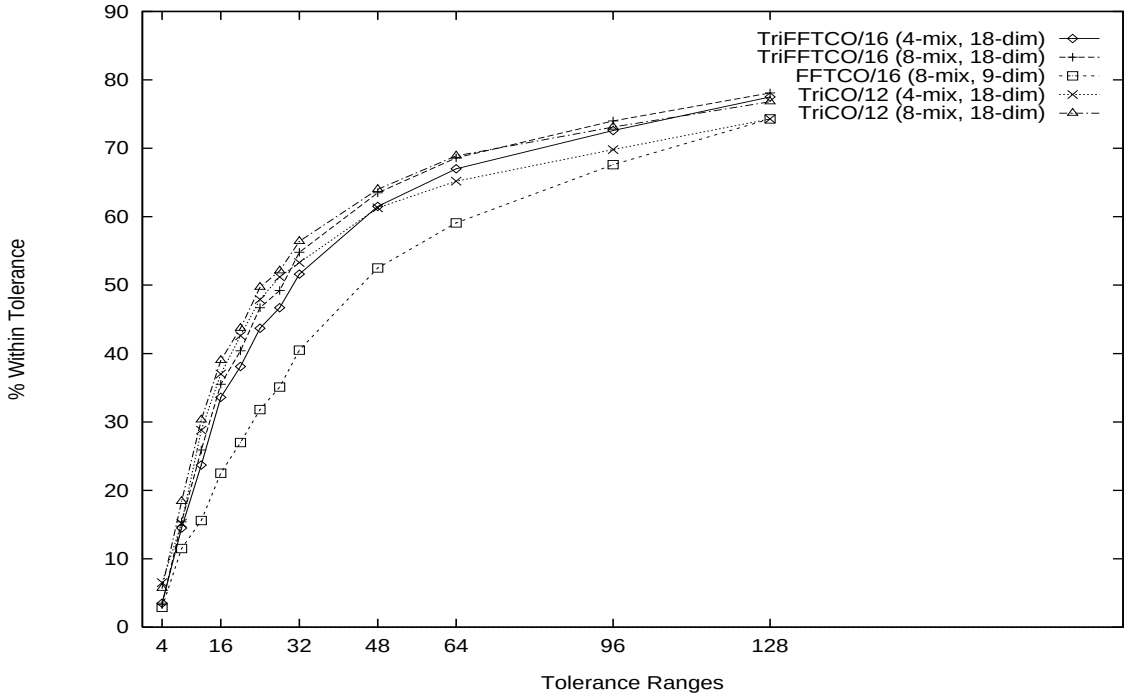

Figure 5.11. FFT Triplicated

results are compared to the TriCO/12 features in Figure 5.12 and Figure 5.13, respectively. As can be seen, for small tolerances the FFT/TriCO/16 features show a slight decrease in performance compared to the TriCO/12 features. The TriFFT/TriCO/16 features, on the other hand, do show a slight improvement in performance. The 4-mixture TriFFT/TriCO/16 uses 512 trainable parameters per state, while the 8-mixture TriCO/12 uses 288, and the 16-mixture TriCO/12 uses 576. As was pointed out earlier, the TriCO/12 features have reached their trainable limit on this data at 8 mixtures and therefore the 16-mixture models do not show the performance increase they might on a larger training set. The TriFFT/TriCO/16 features do show an increase as the number of parameters are increased toward that of the 16-mixture TriCO/12, and therefore do not overtrain as easily. This is most likely due to the inclusion of information from a wider horizontal range by the second layer of triplication.

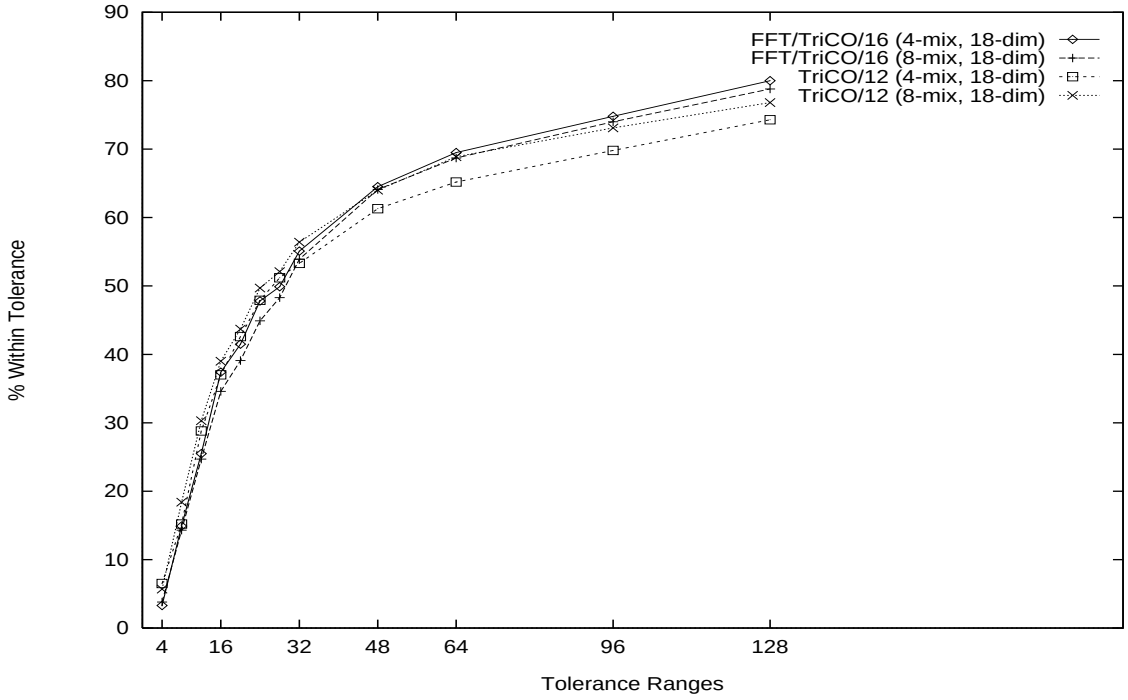

Figure 5.12. FFT of Triplicated Features

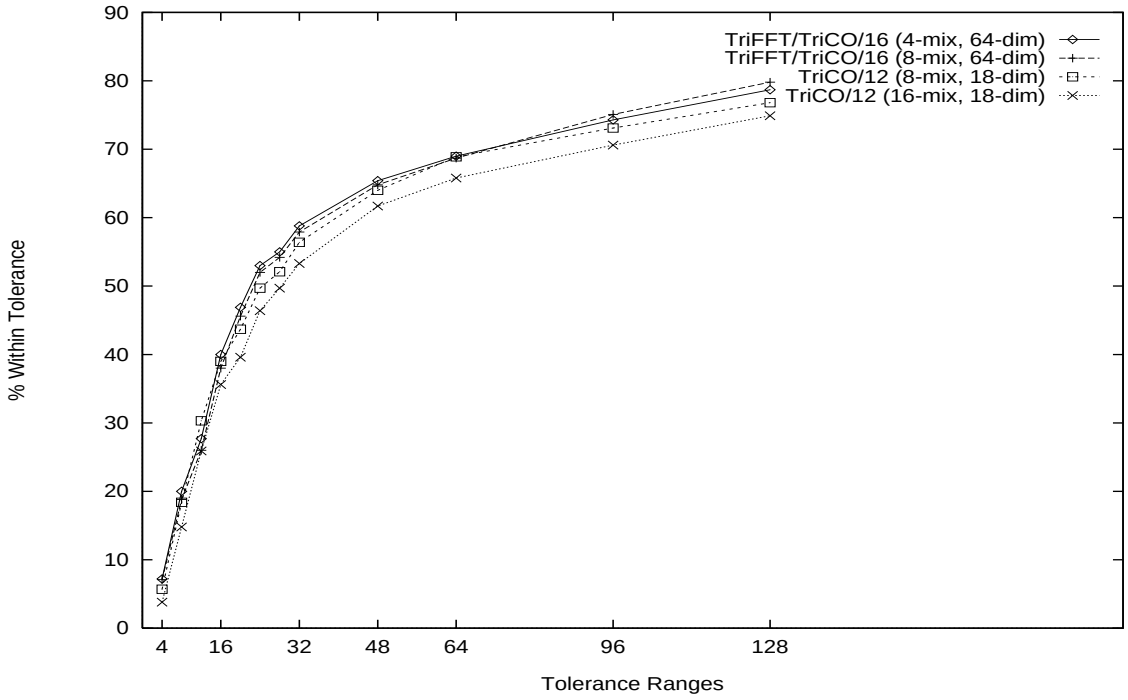

Figure 5.13. Triplicated FFT of Triplicated Features

5.5.7 Contour Direction Features

One set of features that was shown in chapter 4 to produce very good results on discrete characters, was the set of contour direction features developed by Takahashi [1991]. However, the effectiveness of these features on unsegmented data using a sliding window has been previously unknown. Figure 5.14 shows the results of testing models trained using these features (called the CDF88 features here) computed on the contents of a 16 pixel wide sliding window. These 88-dimensional features were then reduced to 32 dimensions by PCA. Window step sizes of 16 pixels and 8 pixels were tested, in which the latter causes windows to overlap by 8 pixels. As shown in Figure 5.14, 8 pixel steps sizes appear to be superior. In comparison with the 18-dimensional TriCO/12 features, these features cannot compete for small error tolerances, however by approximately 32 pixel tolerances, the CDF88 features have caught up to the TriCO/12 features in performance and show significantly better results for tolerances above this.

5.5.8 Multichannel Models

Many of the feature sets shown so far have produced results at a level of performance that is close to our best features, TriCO/12. It is possible, however, that by combining different feature sets, we may be able to achieve a performance level that is greater than that of any one set individually. Attempts to combine the TriCO/12 and Edge features into a single feature vector were unsuccessful as discussed in Section 5.8, and other attempts to combine feature sets may also suffer if the combination is created by concatenation. An alternative to this is to place each feature set into a separate channel. This allows each separate channel to train separately so that there are no interdependencies between the two sets of features.

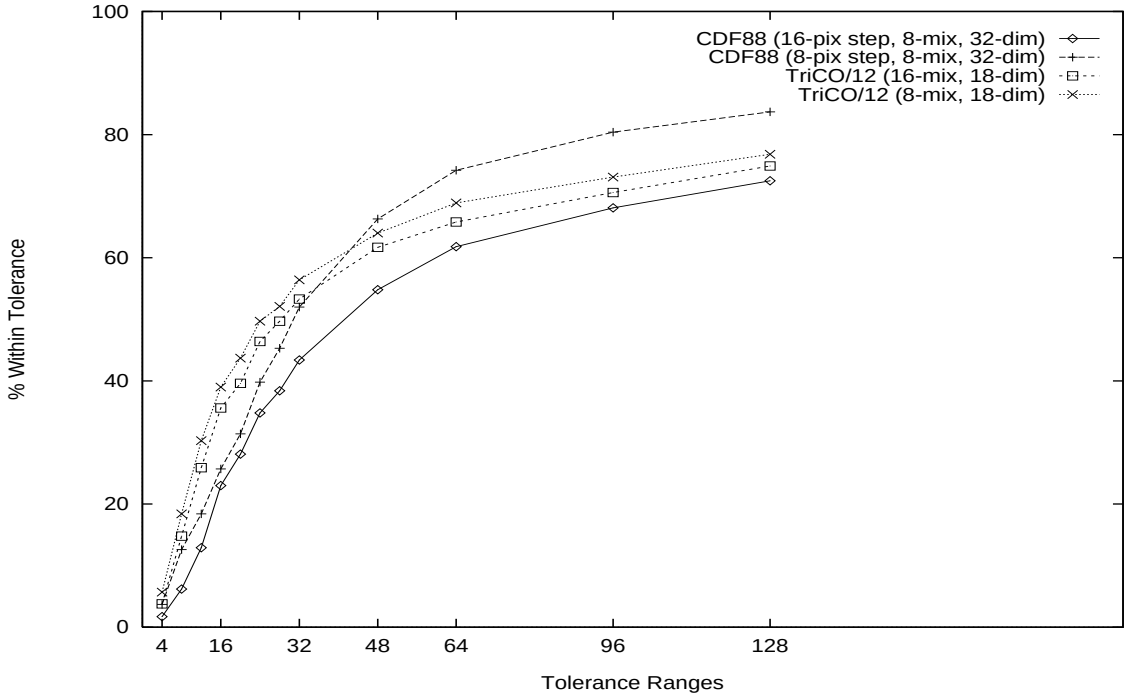

Figure 5.14. Contour Direction Features

In one experiment, 2 channel HMMs were tested using the triplicated CO/12 features in one channel, and the Edge features in a second channel. The TriCO/12 features are reduced to 18 dimensions, while the two edge features are left in their original dimensionality. The results of testing using the N10 grammar are shown in Figure 5.15. The influence each of these channels has on the calculation of forward probabilities (and therefore the overall influence on recognition) is specified by giving each channel a weight. In this experiment, we show results for three different weightings ratios: 25%-75%, 50%-50%, and 75%-25%, where the first weight corresponds to the first channel and the second weight to the second channel. From this graph we see no significant difference between these weightings.

In other experiments, the effects of triplicating the Edge features in the second channel were tested. Figure 5.16 shows these results. In this case, the only significant difference is when the Edge features are weighted more heavily, resulting in a decrease

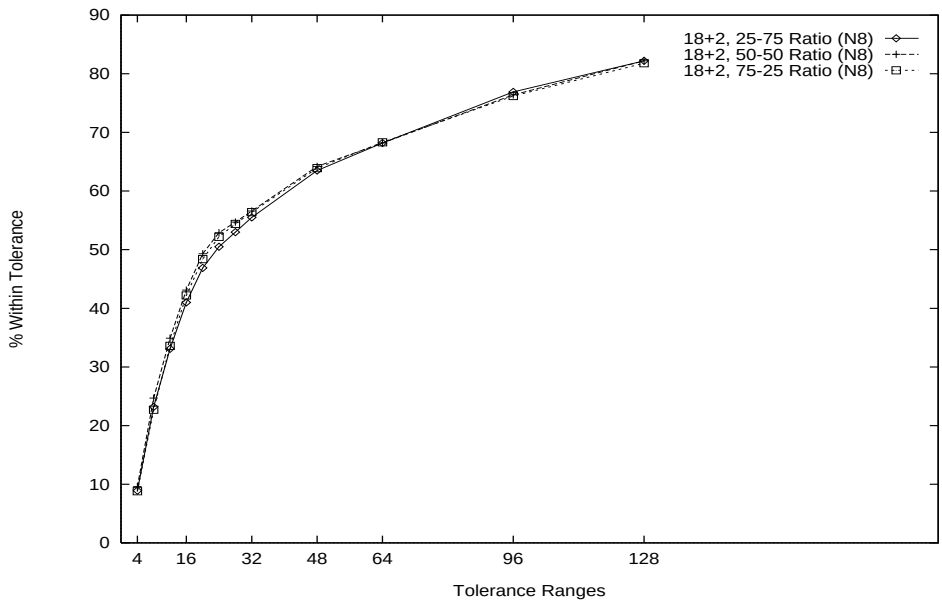

Figure 5.15. Edge Features in 2nd Channel

in performance. A comparison of these two channel Edge feature models to a single channel TriCO/12 model with no Edge features follows in Figure 5.17. The results of using both the N10 and the Forced grammars are presented and both two channel models are shown at a 50-50 weighting of the two channels. Here we see an advantage of using two channels as both of the two channel models outperform the single channel model when the N10 grammar is used. This is particularly interesting since the single channel combination of TriCO/12 and Edge features caused a big performance loss (see Figure 5.8).

Having shown the advantages of using multi-channel models to combine features, three more combined-feature two channel models are presented in Figure 5.18, Figure 5.19, and Figure 5.20. The first of these, Figure 5.18 shows a combined model with 18-dimensional TriCO/12 features in one channel, and 64-dimensional TriFFT/TriCO/16 features in a second channel. Figure 5.19 shows a combined model with 32-dimensional CDF88 features in one channel, and 18-dimensional TriCO/12 feature in a second channel. Lastly, the 2 channel model in Figure 5.20

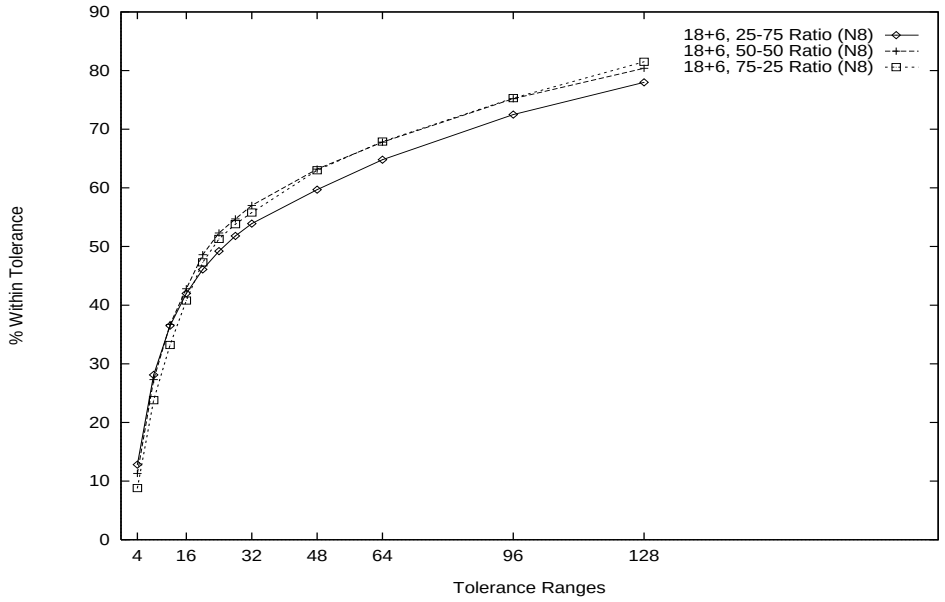

Figure 5.16. Triplicated Edge Features in 2nd Channel

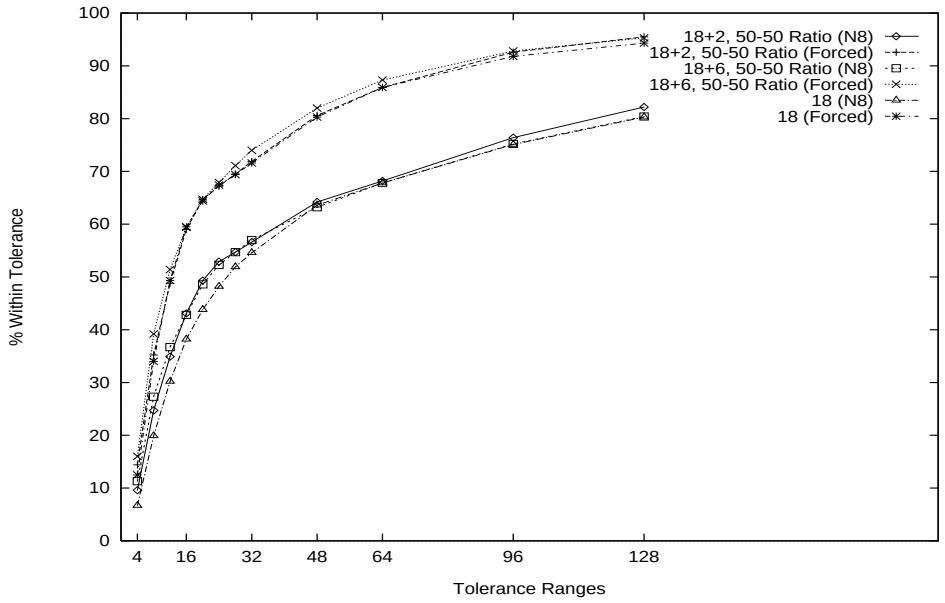

Figure 5.17. Comparison of Two Channel HMMs using Edge Features

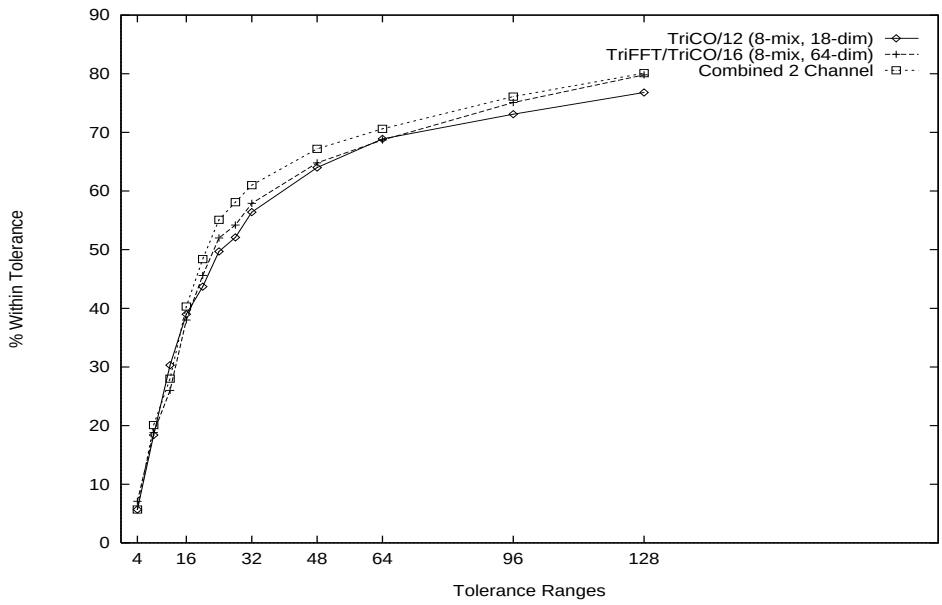

Figure 5.18. Combining CO/12 and FFT Features in 2 Channels

contains 32-dimensional CDF88 features in the first channel, and 64-dimensional TriFFT/TriCO/16 features in the second channel. All three of these examples demonstrate the improvement in accuracy that can be obtained by combining two feature sets in a multi-channel model. In particular, Figure 5.19 and Figure 5.20 each show examples of two feature set in which the results of one set overtake the results of the other as the tolerance range is increased. The combined two channel model in each case obtains results that are close to the best results of the either of the two feature sets over the entire tolerance range. In fact, the combined model of 5.20 even beats the best results of both for the smaller tolerance ranges. This type of performance is also seen in Table 5.18 in which the combined two channel model is always as accurate, but often slightly better, that either of the two feature sets by themselves.

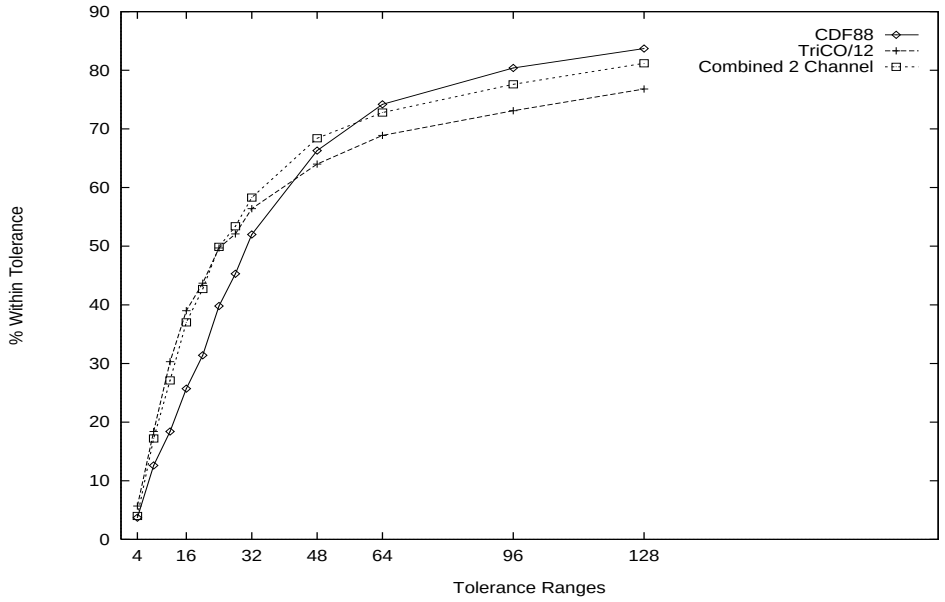

Figure 5.19. Combining CDF88 and CO/12 Features in 2 Channels

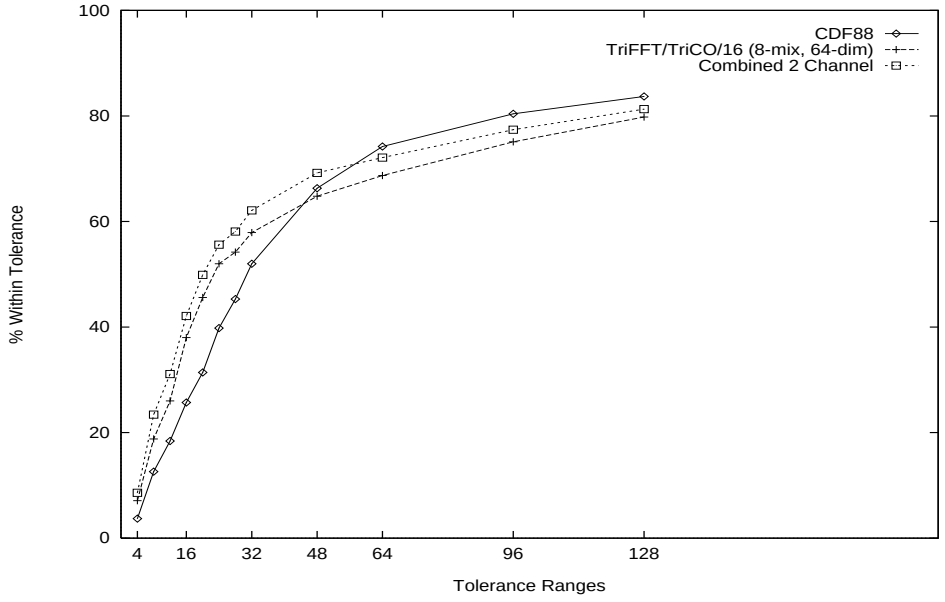

Figure 5.20. Combining CDF88 and FFT Features in 2 Channels

5.5.9 Effects of the Language Model

In the interest of testing the contribution of the language model to the system, a number of tests were performed on the same models, in which characteristics of the language model were varied. Figure 5.21 compares the results of testing using a grammar, and using no language model at all. This figure shows how, without the aid of a grammar, the accuracy of the system is severely degraded. When using the grammar, a network of possible phrases is constructed by tying models together end to end. For purposes of calculating phrase likelihoods, transitions from one model to the next have probabilities set as $\frac{1}{N_{SUCC}(i)}$, where $N_{SUCC}(i)$ is the number of successors to the post-transition model (Young, Woodland and Byrne [1993]). As an alternative, bigram statistics can be computed on the label files for the training images, providing the transition probabilities between any two models based on the observed properties of the data. The performance of the system using the grammar with the addition of bigram probabilities, and the performance using the bigram probabilities with no grammar (any chain of models is possible with probability specified by the bigram statistics), are shown in Figure 5.21 as well. It can be seen in this graph that when no other postprocessing is available, the bigram probabilities contribute quite a lot to the performance of the system. On the other hand, with the presence of the grammar, the addition of these probabilities does not appear to improve system performance greatly. This may be due to the fact that, with only 10 categories and a very tight grammar, the possible sequence choices are already too greatly restricted for these bigram probabilities to be of much use.

5.6 Classification Experiments

Classification results were found to correlate very well with the results obtained using the segmentation measure of the previous experiments. Table 5.6 shows the results

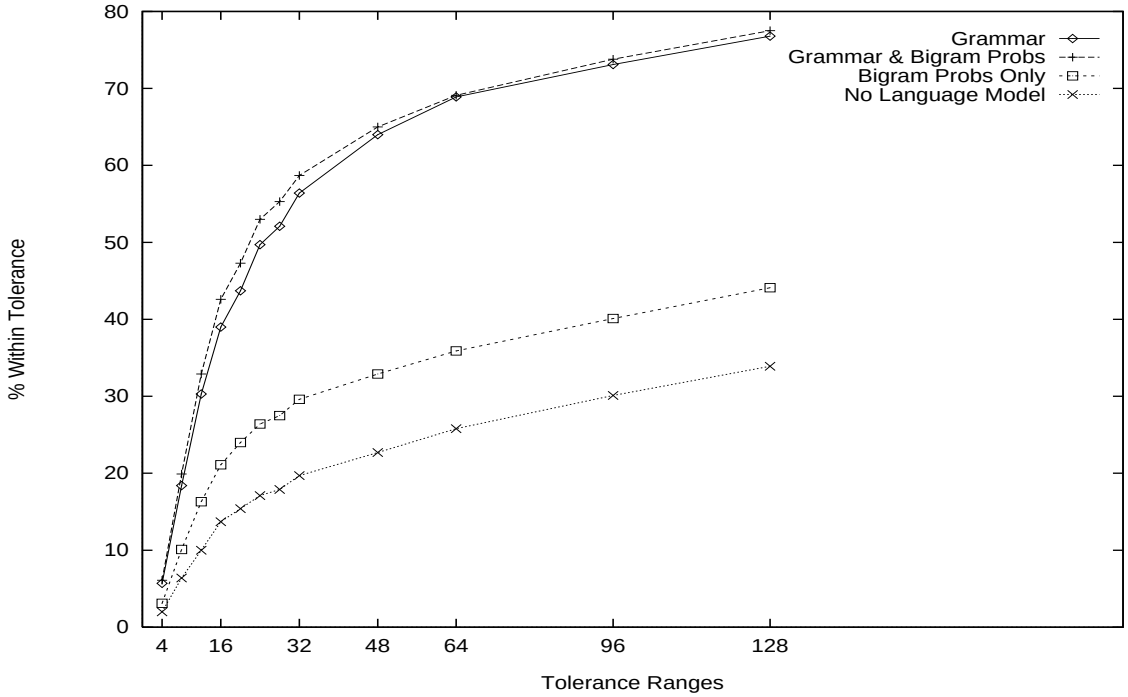

Figure 5.21. Using Bigram Probabilities

of a number of experiments conducted on this recognition task using the grammar shown in Figure 5.5. Two different feature sets were evaluated: 12-dimensional feature vectors triplicated then reduced to 18 dimensions (TriCO/12), and 6 dimensional features triplicated then rotated down to 12 (TriCO/6). In some experiments slant correction was used (marked with *sl* for empirical range slant detection, or **sl** for content zone slant detection) while in others the vectors were computed from the images without any slant normalization. Different training set sizes were used depending on the availability of data at the time.

The best results presented here are for the 16 mixture TriCO/12 trained on 19063 examples. By comparing the results for the 4, 8, and 16 mixture models trained on a set of size 10598 (see Table 5.7), we can see that the over-fitting effects that were seen in the 16 mixture TriCO/12 models of the X model experiments are not present at 16 mixtures in the new set of models. In fact, as the training set size increases,

Table 5.6. 30 Category Classification Results

Feat (Rot Dim)	Mixtures	Train(Test) set	Word Corr	Insert rate	Field Corr
TriCO/12 (18)	4	5525 (1018)	40.12%	13.55%	16.00%
TriCO/12 (18)	8	5525 (1018)	42.42%	12.22%	17.78%
TriCO/6 (12) *sl*	4	6920 (1052)	43.53%	17.7%	16.92%
TriCO/12 (18) *sl*	4	6920 (1052)	43.53%	16.95%	17.11%
TriCO/6 (12)	4	10598 (1030)	39.68%	6.62%	16.31%
TriCO/6 (12) *sl*	4	10598 (1030)	40.79%	5.46%	19.32%
TriCO/12 (18)	4	10598 (1030)	43.21%	6.91%	18.16%
TriCO/12 (18) *sl*	4	10598 (1030)	42.24%	5.99%	18.93%
TriCO/12 (18)	8	10598 (1030)	45.29%	5.17%	21.46%
TriCO/12 (18) *sl*	8	10598 (1030)	45.43%	5.12%	22.52%
TriCO/12 (18)	16	10598 (1030)	45.87%	4.05%	23.01%
TriCO/6 (12)	4	19063 (1018)	41.91%	5.21%	19.65%
TriCO/6 (12)	16	19063 (1018)	45.18%	4.42%	22.30%
TriCO/12 (18)	16	19063 (1018)	48.86%	4.51%	25.74%
TriCO/12 (24) **sl**	16	19063 (1018)	50.54%	5.78%	25.93%

such as in the last two examples of Table 5.7, the performance gain that is achieved by increasing the number of mixtures from 4 to 16 is increased as well.

Table 5.7. Increasing the Number of Mixtures

Feat (Rot Dim)	Mixtures	Train(Test) set	Word Corr	Insert rate	Field Corr
TriCO/12 (18)	4	10598 (1030)	43.21%	6.91%	18.16%
TriCO/12 (18)	8	10598 (1030)	45.29%	5.17%	21.46%
TriCO/12 (18)	16	10598 (1030)	45.87%	4.05%	23.01%
TriCO/6 (12)	4	19063 (1018)	41.91%	5.21%	19.65%
TriCO/6 (12)	16	19063 (1018)	45.18%	4.42%	22.30%

Another interesting comparison is the difference between the TriCO/12 and the TriCO/6 features. Since the TriCO/12 features utilize 50% more parameters than the TriCO/6 features, it is not surprising to find a significant increase in the word correct and field correct rates for the TriCO/12 features such as we find in Table 5.8. What is interesting is that the insertion rate does not decrease, in fact it increases.

This is not by a large amount, but is consistent for different training set sizes. In contrast, increasing the parameters by adding more mixtures consistently decreases the insertion rate.

Table 5.8. TriCO/12 vs. TriCO/6

Feat (Rot Dim)	Mixtures	Train(Test) set	Word Corr	Insert rate	Field Corr
TriCO/6 (12)	4	10598 (1030)	39.68%	6.62%	16.31%
TriCO/12 (18)	4	10598 (1030)	43.21%	6.91%	18.16%
TriCO/6 (12)	16	19063 (1018)	45.18%	4.42%	22.30%
TriCO/12 (18)	16	19063 (1018)	48.86%	4.51%	25.74%

The effects of slant correction are shown in Table 5.9. Feature sets labeled with *sl* have been slant corrected using slants obtained by from empirical zones, while the feature set marked with **sl** has been corrected using a slant obtained from the content zone. Other sets have not been slant corrected. As can be seen, the effects of empirical zone slant correction are not very significant on either the 4 mixture or 8 mixture TriCO/12 models, while the 4 mixture TriCO/6 models show a field correct gain of 3%. One thing that is apparent is that the effects of slant correction on performance are a small but consistent gain in accuracy.

Alternatively, the content zone slant correction caused the TriCO/12 features to be correlated much differently. In order to retain the same amount of information (indicated by the eigenvalues) a reduction down to 24 features was required instead of 18. This increase in the feature dimensions was accompanied by an increase in the insertion rate as was found when the dimensionality was increased by switching from TriCO/6 features to TriCO/12 features. The field correct rate did not change significantly by the introduction of this slant correction, however the word correct rate had its largest increase of all slant correction experiments (albeit, only 2% which is still not very significant). Since the content zone slant requires a lengthy two-pass

detection, the small accuracy gain obtained over the empirical zone slant may not be justified.

Table 5.9. Slant Correction Results

Feat (Rot Dim)	Mixtures	Train(Test) set	Word Corr	Insert rate	Field Corr
TriCO/6 (12)	4	10598 (1030)	39.68%	6.62%	16.31%
TriCO/6 (12) *sl*	4	10598 (1030)	40.79%	5.46%	19.32%
TriCO/12 (18)	4	10598 (1030)	43.21%	6.91%	18.16%
TriCO/12 (18) *sl*	4	10598 (1030)	42.24%	5.99%	18.93%
TriCO/12 (18)	8	10598 (1030)	45.29%	5.17%	21.46%
TriCO/12 (18) *sl*	8	10598 (1030)	45.43%	5.12%	22.52%
TriCO/12 (18)	16	19063 (1018)	48.86%	4.51%	25.74%
TriCO/12 (24) **sl**	16	19063 (1018)	50.54%	5.78%	25.93%

5.7 Summary

In this chapter, the results of a number of experiments involving legal amount recognition have been presented. Many of these experiments have been evaluated using a measure of segmentation, in which the content words were combined into a single model. Later experiments have shown that this measure corresponds well with the classification accuracy, without requiring as much data (and therefore time) to train.

The effects of varying the size of the sliding window were tested in which an optimal size of 8 pixels was found for the 240 dpi check data. In order to include information about how the observations change over time, experiments were conducted in which feature vectors were created as a linear combination of features from three consecutive windows. Poor results were obtained when this linear combination took the form of a lowpass filter, or a calculation of the derivative. However, by concatenating the three vectors together (triplication) and allowing principal component analysis to decide what the relevant components of this linear combination are, a big

increase in accuracy was obtained. The effects of increasing the range of the linear combination to five or seven windows were examined, with poor results in both cases.

A simple set of features which track the height of the upper and lower contour of blackness from window to window encoded a great deal of information for just two features. However, when this was combined with the triplicated vectors and principal component analysis applied, the overall effect was a loss in accuracy.

Another method of feature extraction was attempted in which each window of the CO/xx features were treated as a signal that changes as the window is scanned from top to bottom, and the Fourier transform of that signal was computed. However, these features produced slightly worse results than the CO/xx features by themselves. Other experiments were run in which the Fourier features were triplicated, the Fourier transform was applied to the triplicated CO/xx features, or both. The combination of triplication both before and after applying the Fourier transform was the only experiment that produced results than could not be obtained by other feature sets. This is most likely due to the inclusion of information from a wider horizontal span of the image, caused by the two layers of triplication.

Contour direction features, whose effectiveness was previously assumed to be limited to presegmented characters, worked surprisingly well with a sliding window. Although their overall effectiveness was not as good as our best feature set (CO/12 triplicated and reduced to 18 dimensions), the results were promising. Using a combination of these two feature sets in a two channel model, we were able to obtain an accuracy level better than either set by itself.

The effects of using bigram probabilities estimated from the data were examined. It appears that for such a small lexicon as was used in the segmentation experiments, the added benefit of these bigram probabilities are minimal.

Finally, classification experiments have shown that our best results (obtained using the triplicated CO/12 features) give a word recognition rate of 50.54%, with a field

correct rate of 25.93%. The additional data available for these experiments has shown that as more training data is added, the benefits of increasing the number of mixtures continue to be significant. Additionally, it appears that slant correction can be useful in that it provides a consistent increase in recognition accuracy. However, the small gain achieved is probably not enough to justify the superior, yet lengthy two-pass slant detection performed on the legal amount zone.

CHAPTER 6

Conclusions

The main difficulty in cursive handwriting recognition is the fact that character segmentation points are hard to find. The strength of the Hidden Markov Model is in its ability to perform segmentation and recognition at the same time. In Chapter 4, the effectiveness of a single state Hidden Markov Model for the recognition of discrete, presegmented characters was compared to two other techniques, Learning Vector Quantization and multilayer perceptrons. It was shown that the HMM is more accurate at recognizing these characters than LVQ, and is approaching the performance of the MLP. These results are very promising since the HMM should have a distinct advantage over the MLP when the data is not presegmented.

When creating a model, the amount of training data that is available will limit the number of parameters that can be trained. Since training data is always limited, the distribution of these parameters within the model should be optimized such that the best recognition accuracy is attained. In Chapter 4, we have shown for our data that when the number of states in a model are less than the length of the observation sequence, it is more beneficial to increase the number of states in the model than to increase the number of Gaussian mixtures used to model each state's observation density. Additionally, the use of a single full covariance matrix Gaussian to model the distribution of a state's observations was shown in many cases to be inferior

to the use of multiple diagonal covariance Gaussians, in which a greater recognition accuracy was achieved while requiring less parameters. In an attempt to better utilize data that has similarities between different models or states, the effects of using tied mixture models were explored. However, when compared to untied models using a similar number of parameters, it was apparent that there is a great disadvantage to this type of parameter tying in our domain.

Chapter 5 presented a comparison of a large number of feature sets applied to the task of legal amount recognition using word modeling. A performance measure was presented that is based on a model's ability to segment the check horizontally, defining the area that contains the *content* words. This measure corresponds well to the feature set's ability to perform word-level classification, and requires much less data to train since many word categories are combined into a single 'X' model. Of the feature sets compared, the best results (50.54% word-level recognition, 25.93% field-correct) were obtained using a horizontally sliding window of width 8 pixels. The height of the window is reduced using a non-linear subsampling technique, and a blackness count is taken for each of these 8 pixel wide rows. Relationships from one window to the next are represented by computing a linear combination of the features from three consecutive windows, in which the coefficients are chosen by principal component analysis. Although the accuracy of this system seems low, with the extremely large number of checks processed daily (on the order of 100 million), there can be great benefit in automating even a limited amount of this process. It should also be stressed that the data used in these experiments are very representative of the type of data that is found in the real world, and therefore are a better indication of the results attainable by a production system than results produced using a database of fake checks. Additionally, these results do not represent the maximum accuracy achievable by this system. Rather, the goal here has been to provide a fair comparison of different feature sets without focusing on obtaining their highest possible degree of accuracy,

since this would require much tuning, and therefore a great deal of time for each set of features.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. O. Agazzi and S. Kuo, *Pseudo Two-Dimensional Hidden Markov Models for Document Recognition*, AT&T Technical Journal, September/October 1993a.

2. O. E. Agazzi and S. -S. Kuo, *Hidden Markov Model Based Optical Character Recognition in the Presence of Deterministic Transformations*, Pattern Recognition, 26 (1993b), pp. 1813–1826.

3. M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press, Inc., New York, 1973.

4. L. R. Bahl, P. F. Brown, P. V. deSouza, and R. L. Mercer, *Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition*, ICASSP '86, Tokyo, Japan, 1986.

5. L. R. Bahl, P. F. Brown, P. V. deSouza, and R. L. Mercer, *A New Algorithm for the Estimation of Hidden Markov Model Parameters*, IEEE International Conference on ASSP, April 1988.

6. L. R. Bahl, F. Jelinek, and L. R. Mercer, *A Maximum Likelihood Approach to Continuous Speech Recognition*, IEEE Trans. Pattern Analysis and Machine Intelligence, 5 (1983), pp. 179–190.

7. L. R. Bahl, P. F. Brown, P. V. deSouza, and R. L. Mercer, *Speech recognition with continuous-parameter hidden Markov models*, in Readings in Speech Recognition, A. Waibel and K-F. Lee, eds., Morgan Kaufmann, 1990, pp. 332–339.

8. L. R. Bahl and F. Jelinek, *Decoding for Channels with Insertions, Deletions, and Substitutions with Applications to Speech Recognition*, IEEE Transactions on Information Theory, IT-21 (July 1975).

9. J. K. Baker, *Stochastic modeling for automatic speech understanding*, in Readings in Speech recognition, A. Waibel and K-F. Lee, eds., Morgan Kaufmann, San Mateo CA, 1990, pp. 297–307.

10. L. E. Baum, *An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov processes*, in Inequalities, 3, 1972, pp. 1–8.

11. E. J. Bellegarda, J. R. Bellegarda, D. Nahamoo, and K. S. Nathan, *A Probabilistic Framework for the Recognition of On-line Handwriting*, 3rd International Workshop on Frontiers in Handwriting Recognition, Buffalo, NY, May 1993.

12. J. R. Bellegarda, D. Nahamoo, K. S. Nathan, and E. J. Bellegarda, *Supervised Hidden Markov Modeling for On-line Handwriting Recognition*, 1994 ICASSP, 5 (April 1994), pp. 149–152.

13. R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

14. R. Bellman and S. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, 1962.

15. R. M. Bozinovic and S. N. Srihari, *Off-line Cursive Script Word Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11 (1989).

16. H. Bunke, M. Roth, and E. G. Schukat-Talamazzini, *Off-line Recognition of Cursive Script Produced by a Cooperative Writer*, Proc. 12th Int. Conf. on Pattern Recognition, Jerusalem, 1994.

17. R. G. Casey, *Character Segmentation in Document OCR: Progress and Hope*, 4th Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, 1995.

18. R. G. Casey and K. Y. Wong, *Document Analysis Systems and Techniques*, in Image Analysis and Applications, R. Kasturi and M. M. Trivedi, eds., Marcel Dekker, New York, N. Y., 1990, pp. 1–36.

19. R. G. Casey, *Moment Normalization of Handprinted Characters*, IBM Journal of Research and Development, September 1970.

20. C. -T. Chen, *System and Signal Analysis*, Saunders College Publishing, 1989.

21. M. -Y. Chen, A. Kundu, and J. Zhou, *Off-Line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (May 1994), pp. 481–496.

22. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

23. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, 1973.

24. R. O. Duda and P. E. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, Communications of the ACM, 15 (1972), pp. 11–15.

25. S. A. Dudani, K. J. Breeding, and R. B. McGhee, *Aircraft identification by moment invariants*, IEEE Trans. Pattern Analysis and Machine Intelligence, C-26 (January 1977), pp. 39–46.

26. ERA, *An electronic reading automaton*, Electronic Eng., April 1957.

27. A. J. Elms, *A connected character recognizer using Level Building of HMMs*, IEEE 12th IAPR International Conference on Pattern Recognition, II (October 1994), pp. 439–441.

28. Y. Ephraim, A. Dembo, and L. R. Rabiner, *A minimum discrimination information approach for hidden Markov modeling*, ICASSP '87, Dallas, TX, April 1987.

29. S. A. Euler, B. -H. Juang, C. -H. Lee, and F. K. Soong, *Statistical segmentation and word modeling techniques in isolated word recognition*, in ICASSP-90, Albuquerque, 1990, pp. 745–748.

30. B. S. Everitt, *Cluster Analysis*, John Wiley & Sons, Inc., New York, 1974.

31. R. F. Farag, *Word-level recognition of cursive script*, IEEE Trans. Comput., C-28 (Feb. 1979), pp. 172–175.

32. J. L. Fisher, S. C. Hinds, and D. P. D'Amato, *A Rule-Based System for Document Image Segmentation*, 10th Int'l Conf. Pattern Recognition (ICPR), Atlantic City, NJ, 1990.

33. G. D. Forney, *The Viterbi Algorithm*, Proc. IEEE, 61 (March 1973), pp. 268–278.

34. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Inc., 1990.

35. P. Gader, M. Mohamed, and J. -H. Chiang, *Segmentation-based Handwritten Word Recognition*, 5th US-Postal Service, Advanced Technology Conf., 1992.

36. M. Gilloux, M. Leroux, and J. -M. Bertille, *Strategies for Handwritten Words Recognition Using Hidden Markov Models*, 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japan, Oct 20-22, 1993.

37. M. H. Glauberman, *Character Recognition for business machines*, Electronics, Feb. 1956.

38. G. H. Granlund, *Fourier preprocessing for hand printed character recognition*, IEEE Trans. Comput., C-21 (February 1972), pp. 195–201.

39. A. Gray and J. Markel, *Distance Measures for Speech Processing*, IEEE Transactions on ASSP, 24 (October 1976).

40. R. M. Gray, *Vector Quantization*, in Readings in Speech Recognition, A. Waibel and K-F. Lee, eds., Morgan Kaufmann Publishers, Inc., 1990, pp. 75–100.

41. D. Guillevic, *Unconstrained Handwriting Recognition Applied to the Processing of Bank Cheques*, Dept. of Computer Science, Concordia University, PhD Thesis (August 1995).

42. Y. Hamamoto, S. Uchimura, K. Masamizu, and S. Tomita, *Recognition of Handprinted Chinese Characters Using Gabor Features*, 3rd International Conference on Document Analysis and Recognition, Montreal, Canada, August 14-16, 1995.

43. R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, 1, Addison-Wesley Publishing, 1992.

44. A. Hashizume, P. S. Yeh, and A. Rosenfeld, *A Method of Detecting the Orientation of Aligned Components*, Pattern Recognition Letters, 4 (1986), pp. 125–132.

45. S. C. Hinds, J. L. Fisher, and D. P. D'Amato, *A Document Skew Detection Method Using Run-Length Encoding and the Hough Transform*, 10th Int'l Conf. Pattern Recognition (ICPR), Atlantic City, NJ, 1990.

46. H. Hotelling, *Analysis of a complex of statistical variables into principal components*, JEP , 24 (1933), pp. 498–520.

47. M. K. Hu, *Visual pattern recognition by moment invariants*, IRE Trans. Infor. Theory, IT-8 (February 1962), pp. 179–187.

48. X. D. Huang and M. A. Jack, *Semi-continuous hidden Markov models for speech signals*, in Readings in Speech Recognition, A. Waibel and K-F. Lee, eds., Morgan Kaufmann, 1990, pp. 340–345.

49. F. Itakura and S. Saito, *Analysis synthesis telephony based on maximum liklihood method*, 6th International Congress of Acoustics, Tokyo, Japan, August 1968.

50. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.

51. A. K. Jain and S. Bhattacharjee, *Text Segmentation Using Gabor Filters for Automatic Document Processing*, Machine Vision and Applications, 5 (1992), pp. 169–184.

52. A. K. Jain, Y. Zong, and S. Lakshmanan, *Object Matching Using Deformable Templates*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 18 (March 1996), pp. 267–278.

53. F. Jelinek, *Self-Organizing Language Modeling for Speech Recognition*, in Readings in Speech Recognition, A. Waibel and K-F. Lee, eds., Morgan Kaufmann, 1990, pp. 450–506.

54. F. Jelinek and R. L. Mercer, *Interpolated estimation of Markov source parameters from sparse data*, in Pattern Recognition in Practice, E. S. Gelesma and L. N. Kanal, eds., North Holland, Amsterdam, The Netherlands, 1980, pp. 381–397.

55. M-P. D. Jolly, S. Lakshmanan, and A. K. Jain, *Vehicle Segmentation and Classification Using Deformable Templates*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 18 (March 1996), pp. 293–308.

56. A. Kaltenmeier, T. Caesar, J. M. Gloger, and E. Mandler, *Sophisticated Topology of Hidden Markov Models for Cursive Script Recognition*, 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japan, Oct 20-22, 1993.

57. W. -Y. Kim and P. Yuan, *A Practical Pattern Recognition System for Translation, Scale and Rotation Invariance*, CVPR '94, Seattle, Washington, June 1994.

58. T. Kohonen, J. Kangas, J. Laaksonen, and K. Torkkola, *LVQ_PAK: A program package for the correct application of Learning Vector Quantization algorithms*, International Joint Conference on Neural Networks, Baltimore, June 1992.

59. A. Kornai, *Formal Phonology*, Outstanding Dissertations in Linguistics, Garland Publishing, Inc., 1995.

60. A. Kornai, *Language Models: Where are the Bottlenecks?*, AISB 94 1st Biennial Workshop Series, Summer 1994.

61. A. Kornai and S. D. Connell, *Statistical Zone Finding*, 13th Int. Conf. on Pattern Recognition, III (1996), pp. 818–822.

62. A. Kornai, K. M. Mohiuddin, and S. D. Connell, *An HMM-Based Legal Amount Field OCR System for Checks*, 1995 IEEE International Conference on Systems, Man, and Cybernetics, Vancouver BC, October 1995.

63. A. Kundu, Y. He, and P. Bahl, *Recognition of Handwritten Word: First and Second Order Hidden Markov Model Based Approach*, Pattern Recognition, 22 (1989), pp. 283–297.

64. S. Kuo and O. Agazzi, *Visual Keyword Recognition using Hidden Markov Models*, IEEE Computer Vision and Pattern Recognition Conference, 1993.

65. L. Lam, S. -W. Lee, and C. -Y. Suen, *Thinning Methodologies - A Comprehensive Survey*, IEEE Transactions on Pattern Recognition and Machine Intelligence, 14 (September 1992), pp. 869–885.

66. J. -S. Lee, *Digital Image Smoothing and the Sigma Filter*, Computer Vision, Graphics, and Image Processing, 24 (1983), pp. 255–269.

67. S-W. Lee and Y-J. Kim, *Multiresolution Recognition of Handwritten Numerals with Wavelet Transform and Multilayer Cluster Neural Network*, 3rd International Conference on Document Analysis and Recognition, Montreal, Canada, August 14-16, 1995.

68. S. E. Levinson, L. R. Rabiner, and M. M. Sondhi, *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition*, Bell System Technical Journal, 62 (April 1983), pp. 1035–1074.

69. S. G. Mallat, *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11 (July 1989).

70. J. Mao, K. Mohiuddin, and A. K. Jain, *Parsimonious Network Design and Feature Selection Through Node Pruning*, 12th Intl. Conf. on Pattern Recognition, Jerusalem, Israel, Oct. 9-13, 1994.

71. K. M. Mohiuddin and J. Mao, *A comparative study of different classifiers for handprinted character recognition*, in Pattern Recognition in Practice IV, E. S. Gelsema and L. N. Kanal, eds., Elsevier Science B. V., 1994, pp. 437–448.

72. S. Mori, C. Y. Suen, and K. Yamamoto, *Historical Review of OCR Research and Development*, Proceedings of the IEEE, 80 (July 1992), pp. 1029–1058.

73. R. Nag, K. H. Wong, and F. Fallside, *Script Recognition Using Hidden Markov Models*, ICASSP '86, Tokyo, 1986.

74. K. S. Nathan, J. R. Bellegarda, D. Nahamoo, and E. J. Bellegarda, *On-line Handwriting Recognition Using Continuous Parameter Hidden Markov Models*, 1993 ICASSP, Minneapolis, MN, April 1993.

75. W. Niblack, *An Introduction to Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1986.

76. L. O'Gorman and R. Kasturi, *Document Image Analysis*, IEEE Computer Society Press, 1995.

77. H. -S. Park and S. -W. Lee, *Large-Set Handwritten Character Recognition with Multiple Stochastic Models*, 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japan, Oct 20-22, 1993.

78. W. Postl, *Detection of Linear Oblique Structures and Skew Scan in Digitized Documents*, 8th Int'l Conf. Document Analysis and Recognition (ICPR), Paris, France, 1986.

79. L. R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, 77 (February 1989), pp. 257–286.

80. L. R. Rabiner and B. H. Juang, *An Introduction to Hidden Markov Models*, IEEE ASSP Magazine, 3 (January 1986), pp. 4–16.

81. J. -C. Simon, *Off-Line Cursive Word Recognition*, Proceedings of the IEEE, 80 (July 1992), pp. 1150–1161.

82. H. Takahashi, *A neural net ocr using geometrical and zonal-pattern features*, 1st Intl. Conf. on Document Analysis and Recognition, 1991.

83. C. C. Tappert, *Adaptive On-Line Handwriting Recognition*, 7th Int. Conf. Pattern Recognition, July-Aug 1984.

84. M. R. Teague, *Image analysis via the general theory of moments*, J. Opt. Soc. Amer., 70 (August 1980), pp. 920–930.

85. C. -H. Teh and R. T. Chin, *On Image Analysis by the Methods of Moments*, IEEE Trans. Pattern Anal. Machine Intell., 10 (July 1988), pp. 496–513.

86. J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, 1974.

87. O. D. Trier and A. K. Jain, *Goal-Directed Evaluation of Binarization Methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 17 (December 1995), pp. 1191–1201.

88. A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimal decoding algorithm*, IEEE Transactions on Information Theory, IT-13 (April 1967), pp. 260–269.

89. J. A. Vlontzos and S. Y. Kung, *Hidden Markov Models for Character Recognition*, IEEE Transactions on Image Processing, 1 (October 1992), pp. 539–543.

90. T. Wakahara, *Shape Matching Using LAT and its Application to Handwritten Numeral Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (June 1994).

91. S-S. Wang, P-C. Chen, and W-G. Lin, *Invariant Pattern Recognition by Moment Fourier Descriptor*, Pattern Recognition, 27 (1994), pp. 1735–1742.

92. J. White and G. Rohrer, *Image Thresholding for optical character recognition and other applications requiring character image extraction*, IBM Journal of Res. dev., 27 (July 1983), pp. 400–411.

93. K. Y. Wong, R. G. Casey, and F. M. Wahl, *Document Analysis System*, IBM J. Research and Development, 26 (November 1982), pp. 647–656.

94. B. A. Yanikoglu and P. Sandon, *Recognizing Off-Line Cursive Handwriting*, CVPR '94, Seattle, Washington, June, 1994.

95. S. J. Young, P. C. Woodland, and W. J. Byrne, in HTK: Hidden Markov Model Toolkit V1.5 Reference Manual, Entropic Research Laboratories Inc., December 7, 1993.

96. B. Yu and A. K. Jain, *A Robust and Fast Skew Detection Algorithm for Generic Documents*, to appear in Pattern Recognition, 1996a.

97. B. Yu and A. K. Jain, *A Form Dropout System*, submitted to 13th IEEE Int. Conf. on Pattern Recognition, Vienna, August 1996b.

98. C. T. Zahn and R. Z. Roskies, *Fourier Descriptors for Plane Closed Curves*, IEEE Trans. Comput., C-21 (March 1972), pp. 269–281.

99. A. I. Zayed, *Advances in Shannon's Sampling Theory*, CRC Press, Inc., 1993.