A Novel Approach to Handwritten Character Recognition

by Eddie Clarke, B.Sc.

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy, September 1995.

Abstract

A number of new techniques and approaches for off-line handwritten character recognition are presented which individually make significant advancements in the field.

First, an outline-based vectorization algorithm is described which gives improved accuracy in producing vector representations of the pen strokes used to draw characters. Later, vectorization and other types of preprocessing are criticized and an approach to recognition is suggested which avoids separate preprocessing stages by incorporating them into later stages. Apart from the increased speed of this approach, it allows more effective alteration of the character images since more is known about them at the later stages. It also allows the possibility of alterations being corrected if they are initially detrimental to recognition.

A new feature measurement, the Radial Distance/Sector Area feature, is presented which is highly robust, tolerant to noise, distortion and style variation, and gives high accuracy results when used for training and testing in a statistical or neural classifier. A very powerful classifier is therefore obtained for recognizing correctly segmented characters. The segmentation task is explored in a simple system of integrated over-segmentation, character classification and approximate dictionary checking. This can be extended to a full system for handprinted word recognition.

In addition to the advancements made by these methods, a powerful new approach to handwritten character recognition is proposed as a direction for future research. This proposal combines the ideas and techniques developed in this thesis in a hierarchical network of classifier modules to achieve context-sensitive, off-line recognition of handwritten text. A new type of "intelligent" feedback is used to direct the search to contextually sensible classifications. A powerful adaptive segmentation system is proposed which, when used as the bottom layer in the hierarchical network, allows initially incorrect segmentations to be adjusted according to the hypotheses of the higher level context modules.

Acknowledgements

I would like to thank my supervisors. I am indebted to Professor David Elliman for sharing his endless stream of ideas with me, for his guidance, support and encouragement, and particularly for supporting me financially since my grant ran out. I am very grateful to Dr Edmund Burke for stepping in when it appeared I was to be supervisorless, for his constant nagging to write my ideas down (without which I would still be writing now) and for his diligent proofreading.

My thanks are also owed to the members of the Machine Intelligence Group (MIG) both for their intelligent discussion of ideas and for making the office an enjoyable and entertaining place to work.

I would also like to express my gratitude to all the support staff at the Department of Computer Science for sharing their invaluable time and technical expertise in helping me in my role as MIG system administrator. In particular, I am indebted to William Armitage for teaching me the ropes, rescuing me when I got out of my depth and, most importantly, for giving me somewhere to live.

I am grateful to the Science and Engineering Research Council (now EPSRC) for its financial support which enabled me to undertake this research.

Finally, I would like to thank my family and friends for their constant assistance, support and encouragement throughout my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the difficult problem of identifying handwritten characters using a computer. The characters are presented to the computer in an electronic representation, produced by an optical scanning device. This representation encodes the shape of characters with no information as to the letters or numerals they represent. The characters are scanned from a pre-drawn image so no information about how they were drawn is available, only the final shape. Identifying such images is called off-line recognition.

The field of off-line character recognition is a large one. It has a great number of applications and its commercial potential is considerable. However, marketable products have so far been limited to machine-printed character recognizers. While optical character recognition (OCR) software for machine-printed characters is commonly found in offices today, systems for recognizing handprinted characters have not quite reached the levels of accuracy required for practical use. Systems for recognizing handwritten characters are even less accurate. The difficulty of the problem is due to the enormous style variability of handwritten characters.

Character styles vary in many ways: relative size, aspect ratio, sharpness of angles, slant, slope, ornamentation, and the number, ordering, position, direction and thickness of strokes. Factors causing these variations have been shown to include the writer's age, health, handedness, education, profession, nationality, and in specific cases the speed of writing, method of presentation and motivation for writing. The writing implement used and quality of the writing surface also causes style variation. (Kuklinski [Kuk84])

This thesis presents a range of work on various aspects of the off-line handwritten character recognition problem. The main direction of the work is towards a novel approach for incorporating contextual sensibility throughout the recognition process. Attaining contextually sensible recognition is the key to making off-line handwritten OCR accurate enough for practical use.

Off-line character recognition has traditionally focused on the classification of isolated characters, without considering the context in which those characters are found. Although it is difficult to compare recognition results when different test data is used, the field appears to have reached the stage where machine recognizers are very close to achieving the same level of accuracy as humans on isolated handwritten characters.

Estimates for human error rates on handwritten characters without context range from 4% to 12%, and as high as 28% for cursive script [SSK77] [EUF90]. It is generally accepted that human errors on the isolated character recognition task are mainly caused by ambiguous characters. This ambiguity is caused either by corruption of the image by noise (e.g., dirt or poor scanning) or poor drawing (e.g., bad ink flow or an uneven writing surface), or is naturally present (e.g., a vertical line image might be a '1' or an 'l'). The human error therefore closely approximates the degree of overlap of character class boundaries at the underlying image level (rather than at the feature level) which puts a practical limit on isolated character recognition for both humans and machines.

A machine recognizer is therefore limited to roughly the same level of accuracy as humans, for recognition without context. Despite approaching that level of accuracy, noncontextual machine recognition of handwritten and cursive characters is still not accurate enough for practical use. While research continues to improve features and classifiers for isolated recognition, the benefits of these improvements become smaller as accuracy gets closer to the practical limit. If character recognition is to progress to useful levels of accuracy it is essential that contextual information be included in the classification. This idea has been the motivation for the research presented in this thesis.

Attempts have been made to incorporate contextual information into character recognition. However, these attempts have mainly been rather limited, application specific, feedforward correction processes for dictionary checking. No general system for contextual processing has appeared. Chapter 8 of this thesis proposes a flexible, modular, hierarchical network of classifiers for general contextual classification, using feedback as well as feedforward communication to produce contextually sensible results.

Although the proposed hierarchical system is intended to be general in nature, its development requires testing on specific practical problems. The size of these problems and the need for modifications and developments of existing methods for use in the contextual hierarchy have limited the scope of this thesis to one specific task: character recognition. Much of the work presented here is concerned with the development of character recognition solutions which can be incorporated into the contextual hierarchy with a view to testing the proposed system. At the same time, these solutions can be used outside of the hierarchical system and represent significant developments in the field.

After reviewing the fields of character classification and contextual processing in chapter 6, chapter 7 explores a method for contextual recognition of words based on classifications of individual characters. To a large extent this method follows on from previous work on contextual word recognition. The aim of this was partly to develop a stand alone word recognizer but mainly to test the possibility of integrating the contextual recognition with the prior stages of segmentation and isolated character recognition. The method, as it stands, is not robust enough for practical use but it demonstrates the potential of contextual information to find the correct segmentation and produce correct classifications.

Work has also been carried out to improve the recognition of isolated characters. There are two main approaches to the problem: structural and statistical (the neural approach can be seen as a type of statistical approach) and improvements to both of these have been made. Chapter 6 reviews the classification aspects of these approaches, while chapter 2 surveys the earlier stages of processing the isolated characters.

The structural approach involves extracting shape primitives from images and basing classification on the types of primitives and their relationships. The images are usually thinned to a line or vector representation before extraction to simplify the process. The weakness of this approach is that it relies heavily on accurate extraction of the primitives. It is very prone to errors caused by noise and distortion, and also by errors introduced in the thinning process. Chapter 3 presents an improved vectorization algorithm for producing accurate vector representations of character images. This method, developed from a popular existing method, includes significant improvements to the vectorization of "junctions" (areas where pen strokes overlap) and produces vectorizations which are much closer to whole strokes than those output by the previous algorithm.

The statistical approach to isolated character recognition is based on extracting vectors of feature measurements from images. Classification is based on comparing these feature vectors to statistical models of character classes. The main problem with existing feature approaches is that they are not robust enough for practical use. A new feature is presented in chapter 5 which is highly robust, tolerant to noise and distortion, and invariant to position and size, and partially to rotation. The new feature gives high levels of accuracy and generalization on two of the most popular standard character databases. The accuracy and robustness of this feature are essential for the isolated character recognition stage in the proposed contextual hierarchy.

At the segmentation stage of character recognition, two methods are presented. One is a simple algorithm intended only to test the integration of segmentation with contextual classification in chapter 7. The other is a proposal for adaptive segmentation which forms the bottom layer of the hierarchical system. Adaptive segmentation is a system where the initial segmentation is modified during contextual processing to better fit the expected classifications. This overcomes one of the main problems in character recognition — that recognition is restricted by the accuracy of the initial division of the input data. It is suggested that the new feature of chapter 5, can be used in conjunction with an adaptive segmentation system.

The earlier preprocessing stages of character recognition are evaluated in chapter 4 and it is concluded that several of them can be detrimental to recognition and that others can be performed more efficiently and effectively by incorporating them into later stages. The methods and approaches in this thesis attempt to avoid separate preprocessing stages wherever possible.

Another aspect of the contextual hierarchy is a requirement for a reversal of the classification process. Contextual corrections are to be verified by comparing inputs which represent the output class with the actual inputs. This is done at each stage, allowing a novel feedback mechanism to direct the search for the correct classification. This requirement is considered for each stage of the recognition process and several methods presented here are designed to be capable of producing such representatives. The new features can produce representatives of the original character shape; the classifier used in chapters 5 and 7 can produce representative feature vectors for each class and the word recognizer can produce representative characters for the output word.

This thesis therefore presents an accurate vectorization algorithm, a powerful and robust feature for isolated character recognition and a word recognition approach using multiple segmentations which are all significant improvements on existing methods. In addition, it proposes a powerful new approach for further research into off-line handwritten character recognition and lays the groundwork for its future testing and development. It is hoped that a full implementation of the contextual hierarchical system, using the ideas and techniques which have been developed here, will lead to a highly accurate recognition system which is robust and powerful enough for practical application.

Chapter 2

Off-Line Character Recognition

2.1. Summary

This chapter reviews the research field of off-line character recognition. Section 2.2 describes the overall problem and the initial forms of the input data. Sections 2.3 to 2.5 review work on the typical early processing stages: preprocessing of the image representation to aid subsequent recognition, segmentation of text into isolated characters and extraction of features which exhibit the distinctive characteristics of character classes. Section 2.6 discusses character databases for recognition experiments. Conclusions on the current state of this part of the research field are made in section 2.7. Section 2.8 describes the notation and symbols used in this chapter.

A review of the actual recognition stage of off-line character recognition (classification and contextual processing) is presented in chapter 6 as it seems a logical division and does not concern the early chapters of this thesis. Note, however, that an ideal division, even within the sections of this chapter, is impossible as the different stages of character recognition often overlap, for example, normalization (which is usually considered to be a form of preprocessing) is sometimes performed after feature extraction [DL90a], and segmentation can be integrated with recursive contextual processing [TA91].

2.2. Outline of the Problem

In off-line character recognition, an image is scanned and a digital representation of the image is stored. An attempt is then made to identify the text in the representation, translating it into a machine-readable format, usually ASCII.

2.2.1. The Image

The image could be any document containing characters. Typical images are engineering drawings, computer programs, accounts sheets, transaction statements, envelopes, forms, cheques, mail, maps and airline tickets. The image is not necessarily a pure sample of writing or print. Many images used in real world applications will have additional lines, e.g., boxes on a form, or noise and obscurations, e.g., dirt, smudged ink. Drawings, pictures and diagrams may also be present in an image. It may not be readily determinable which areas of the image are text, to be recognised, and which are not. Even if found, the text may not always be in the desired orientation for recognition.

The need for correction of the orientation of the digital representation of a document can be aided, reduced and in some cases eliminated before scanning by using special marks on forms. This may be by including distinctive, easily identified ink markings on the form or by physically marking the form, e.g., cutting off a corner. Physical marks may allow forms to be correctly orientated mechanically for automated presentation to the scanning device.

The process of identifying where the writing lies within an image is called text segmentation. It is frequently the first step in character recognition applications. Many methods exist for performing this segmentation, usually from a greyscale scan (see section 2.2.2.2), e.g., pixel classification (region growing and splitting) [PR78], brightness thresholding [WNR74] [WR83] [HS88], edge detection [Dav75] [Per80] [TP86] [SB91], relaxation [HZ83], texture analysis [CP79] [WWC82] and connected-component analysis [FK88].

This thesis is not concerned with the task of text segmentation and for the most part does not deal with orientation correction. For the problems on which this work concentrates, it will be assumed that text segmentation has already been performed. In reality, the images used in this research contain only text, and the correct orientation is already known.

2.2.2. The Digital Representation

The image is optically scanned and digitized into a numerical representation suitable for input to a digital computer. The numerical representation describes the positional distribution of optical brightness measurements, quantized into a grid of square elements, *pixels* (also called *image elements*, *picture elements* or *pels* [GW77]). Square pixels are used because rectangular ones complicate invariant feature extraction; rectangular pixel lengths of part of a character will vary according to its orientation. The scan covers the whole document by either moving a scan head over a stationary document or by moving the document in front of the scan head using a rotating drum and stepped friction feed [PS80].

The optical brightness is a measurement of light reflected off the document. The most common types of scanning device measure this using either a Charged Coupled Device (CCD) or a photomultiplier tube. The photomultiplier produces a digital output proportional to the optical density of its input. The CCD produces a similar output in the form of a wave, with voltage proportional to the illumination, which is then passed through an analogue-to-digital converter. The photomultiplier is used in laser scanners; it records the brightness of a fine laser beam reflected off the document. With CCD devices, the illuminated image is presented to the CCD via a camera or a fibre optic pipe.

The exact format of the digital representation of the image depends on the software used with the scanner. The following sections discuss commonly used formats.

2.2.2.1. Bi-Tonal Representations

A standard file format is the bitmap, where positions in the representation correspond to positions in the image, and '0's represent white pixels and '1's represent black pixels. The bitmap is therefore only capable of storing a bi-tonal (black and white) representation of the image. Figures 2.1a and 2.1b illustrate a simple image and its corresponding bitmap representation.

The bitmap is an elementary format in the field of character recognition. Though many different formats exist, they consist of the same basic information. Variations exist where the pixel value is encoded in a whole byte as ASCII decimal (more easily read by humans), rather than in a single bit (much more compact). Other formats vary in the compression method, or the header information, e.g., height and width of the image, resolution and labels. Despite the simplicity of the bitmap, there are a large number of variant formats. Platform-specific, printer-specific and application-specific bitmap standards are abundant, e.g., Epson, HP LaserJet, Printronix, MacIntosh MacPaint format, Sun icon format, the Atari ST Degas .pi3 format, BBN BitGraph terminal Display Pixel Data (DPD), MGR, CMU window manager, X10, X11 and Zinc. Fortunately, since all these black and white image representations contain essentially the same data, converting between formats is not a problem. However, for the purposes of designing a character recognition system, it is desirable to work from a platform-independent bitmap standard. A popular choice is the Portable Bitmap Format (PBM).

This research has used run-length format (also called chord encoding) as a simple, fairly compact, standard for bitmap storage. Rather than record each pixel individually, the run-length format records the length of runs of black or white pixels on a line. Since black and white must alternate in a bitmap representation, there is no need to indicate the colour to which the length refers, provided a convention exists to define the starting colour. Figure 2.1e illustrates a run-length encoding of the bitmap in figure 2.1b.

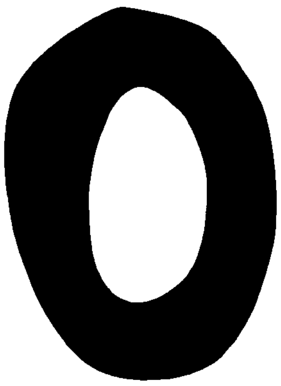
Although bitmaps are commonly used, they have the weakness that images are not always clearly black and white. Pixels which are not clearly one colour or the other may be desired parts of the image, or may be unwanted noise. This is particularly problematic when light inks or coloured paper are used. Many scanners produce output in the form of grey levels or colour, thus providing more information on which to base recognition decisions.

2.2.2.2. Greyscale Representations

The next step up from bitmap representations are greyscale representations, also called greymaps. These give more information than a bitmap by encoding the colour of a pixel on a scale of "greyness." '0' represents white and some maximum value represents black. As in the bitmap format, the positions of these "grey level" numbers in the representation correspond to positions in the scanned image. Figure 2.1c shows the greyscale representation of the image in figure 2.1a.

The greyscale information allows a more flexible handling of potentially noisy areas and is an aid to segmentation. Greyscale is another elementary format, varying mainly in the number of grey levels used. Equivalent standards include the Flexible Image Transport System (FITS), Usenix FaceSaver format, Lisp Machine multi-plane bitmaps and PostScript. Portable Greymap Format (PGM) is a lowest common denominator greyscale file format in the PBM family which is similarly popular as a platform-independent format.

Any greyscale format can be converted to a bitmap by thresholding the grey levels into black or white. This reduces the size of the data and divides an image into a foreground and background. Several surveys of the variety of grey level thresholding techniques have been published. A fairly recent one is by Sahoo *et al.* [SSW88]. The main strategies are *global thresholding*, *dynamic thresholding* and *edge detection*.



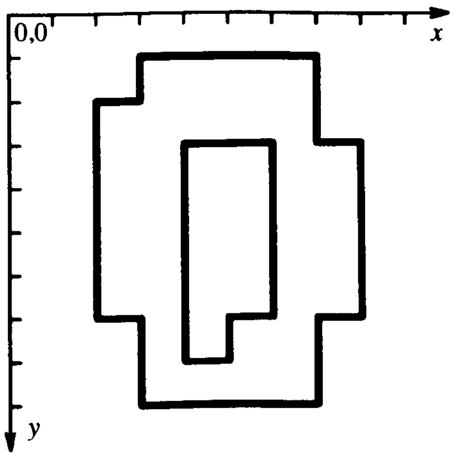
a)

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

b)

0	0	0	0	2	1	0	0	0	0
0	0	2	15	15	15	15	1	0	0
0	0	15	15	13	15	15	4	0	0
0	1	16	14	1	3	16	16	0	0
0	2	16	9	0	1	14	16	1	0
0	1	16	7	0	0	14	16	1	0
0	0	16	8	0	1	15	16	1	0
0	0	4	12	4	10	16	16	1	0
0	0	2	16	16	16	16	2	0	0
0	0	0	1	3	3	2	0	0	0

c)



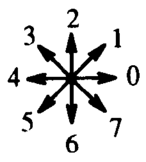
d)

10
3 4 3
2 5 3
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
3 1 1 2 3
3 4 3
10

e)

12 - Number of coordinates
0 - Loop level number
2 2 2 7 3 7 3 9 7 9 7 7
8 7 8 3 7 3 7 1 3 1 3 2
6 - Number of coordinates
1 - Loop level number
4 3 6 3 6 7 5 7 5 8 4 8

f)



Definition of direction numbers

0 - Chain level number
2 2 - Starting coordinate
6666760002122232444S
1 - Chain level number
4 3 - Starting coordinate
1076665532222

g)

Figure 2.1 Example image and possible representations: a) image, b) bitmap, c) greymap, d) outline, e) run-length encoding, f) outline coordinates, g) chain coding.

Global thresholding uses a single threshold grey level: all values below it become white and all those above become black. In practice, local contrast variations mean there is not always a clear grey level division between black and white pixels over the whole image. Dynamic thresholding uses local properties of the image to compute the threshold level. An example of this is Pérez and Gonzales' method [PG87] which is based on a Taylor series expansion. Weszka [Wes86] provides a good survey of dynamic thresholding methods.

Throughout the 1980s much research was carried out on edge detection methods. These attempt to determine where the borders between black and white are in the image. The simplest method is to compute a gradient of image intensity; the Roberts and Sobel methods are among the more common of this kind (see [Nev86]). Surface fitting techniques approximate the image intensity profile with an analytical function and use the computed derivative to determine the edges. For example, Hueckel [Hue73] used a Fourier transform to approximate the image surface; Haralick's facet model [Har84] used sets of masks on the image to calculate a two-dimensional discrete orthogonal polynomial approximation. Surface fitting methods are improved by the use of second-order derivatives, where edges are indicated by zero crossings. White and Rohrer [WR83] used Laplacian gradient operators for this purpose and Marr and Hildreth [MH80] used a Laplacian-Gaussian operator. An optimal clustering method was used by Manohar [Man83]. Kay and Lemay [KL86], Assal *et al.* [AHF88], and Navatia and Babu [NB80] all used linear model schemes. These are template matching methods where a small array of pixels (e.g., 3x3 or 5x5) is passed over the image representation to locally determine the presence of edges.

2.2.2.3. Colour Representations

File formats also exist for colour images, though colour is not essential for many character recognition applications. It is sometimes useful for identifying particular pieces of text within a document. For example, in map recognition, colour may often be used in the text segmentation stage to distinguish words from lines (roads, contours, etc.).

Many colour formats are in use, some of the most common ones being Graphic Interchange Format (GIF), Joint Photographic Experts Group (JPEG) and Tagged Image Format File (TIFF), all of which are platform-independent. A variety of platform-specific formats are widely used, such as Amiga's InterLeaved BitMap, Hold-And-Modify, DynaHAM and

Sliced HAM formats, IBM's PC Paintbrush (PCX), MacIntosh's QuickDraw Picture (PICT), and Atari's Spectrum and Degas .pi1. As with bitmap and greyscale representations, there is a lowest common denominator file format, Portable Pixmap format (PPM). Each pixel colour is represented by three values indicating the red, green and blue content of the colour. The allowed range of these values determines the accuracy of the colour storage. The other colour formats can be accurately converted to PPM format if the colour range of the pixmap is sufficient. PPM format is not generally as popular as PBM and PGM for platform-independent colour representations; some compression, as used in GIF and JPEG for example, is desirable given the larger space requirements of colour images.

Any colour format can be converted to a greyscale format by quantizing the colour spectrum into grey levels. Formats capable of storing colour images are often just used for bitmaps or greyscales — a bitmap being a colour representation with only two colours. One of the scanners used for this research can only produce black and white scans but it produces files in TIFF format. This is because TIFF is a widely used standard and, presumably, because it will make future upgrades to the scanner easier (the basic model could be improved to do colour scans without having to change the output format). Some colour formats have simplifications to enable them to more efficiently store just bitmaps or just greyscales.

2.2.2.4. Outline Representations

One format which is of particular relevance to character recognition is the outline format. Many approaches to the problem find it useful to examine the outline of a character. (The main alternative is to look at a skeleton of the character; this however requires alterations to the raw image representation. This approach will be dealt with in later sections.) Compressed bitmap formats, such as the run-length encoding used in this research, are not well suited to measuring certain features such as the perimeter of a character. The run-length is therefore translated to an outline description prior to processing. Various methods of outline creation have been proposed (Freeman [Fre61], Cederberg [Ced79], Danielsson [Dan82], Kim *et al.* [KLK88], Dinstein *et al.* [DLG91]). This research prefers the method of Elliman and Connor [EC90], which uses the relative position of pixel runs on adjacent lines to create nested outline loops in a single pass. Figure 2.1d shows an outline representation of the image in figure 2.1a which has been extracted from the bitmap in figure 2.1b using Elliman and Connor's method.

The outline of a shape holds essentially the same information as a bitmap so the outline format can be regarded as a bitmap variant. It should be noted, however, that some approaches will start with a pixmap or greyscale representation and use the higher level information to estimate the edge of the shape. This is not exactly the same as quantizing the higher level representation down to a bitmap and storing the outline of that bitmap. For the purposes of this explanation it will be assumed that the outline is being taken from a bitmap.

Depending on the encoding used, the outline is either the line following the black/white pixel boundary or a line joining the midpoints of the black edge pixels. It may be stored in a number of ways, for example, a series of coordinates or a chain code. One compact format suggested by Connor [Con89] is a series of orthogonal vectors indicating the length and direction of the line in alternating horizontal and vertical directions. A related format is chain coding [Fre61]. This uses the fact that pixels on a border are adjacent in one of eight directions to produce a compact coding for the next pixel in sequence. Each of the eight directions is assigned a number (usually 0 to 7). For each border pixel, the direction number of the next pixel around the outline is written in order. The coding corresponds to a chain of short vectors from one boundary pixel to the next. Chain codes are commonly used in character recognition as they are rotationally invariant, simple to extract certain features from (e.g., outline perimeter length) and can be used to detect certain structural characteristics (e.g., corners [Fre74] [Ced79]).

Figure 2.1f shows the outline encoding used in this research which consists of ordered, nested loops (see below) and coordinate points. The encoding is of the outline in figure 2.1d. Figure 2.1g shows a chain coding of the black pixel edges of the bitmap in figure 2.1b.

These methods assume certain simplifications about the image that apply to bitmaps but not necessarily to later stages of preprocessing. After smoothing, for example, the outline may no longer be a sequence of horizontal and vertical lines. The points on the outline may no longer be at integer coordinates and the outline may no longer follow the pixel boundary. Sometimes, after preprocessing, the outline is not even represented by straight lines though this is more common when dealing with diagrams and engineering drawings (where it is desirable to recognize circles and ellipses) than with text. It is useful for the format to be able to describe the image at later stages of processing in addition to the initial scanner output. This research uses a less compact outline coding: a list of coordinate points representing the ordered corners of the outline. This format has the capability to describe outlines with floating point coordinates if required.

The only complication caused by the outline format is that, when extracted from the original pixel boundary, the outline path for a given bitmap may depend on whether pixels are considered to be 4-connected or 8-connected. This causes differences in the number of outline loops in the representation. Also, pixels belonging (or not belonging) to a character may be incorrectly disconnected from (or connected to) the rest of that character outline. These problems should not be considered to be errors. The assignment of outlines, or parts of outlines, to individual characters is a segmentation problem and not a requirement of the basic representation.

The outline representation may not be determined until after preprocessing, or the determination may be combined with the preprocessing stages. It is, however, more common to establish the outline representation at the outset as this frequently simplifies the preprocessing (there are fewer points on an outline than in an equivalent bitmap).

Beyond simply storing the outline representation of the black/white pixel boundary, it is useful at this stage to preserve relationships between the outline loops. Some outline loops will be contained in other loops, i.e., they are holes in the image. Such loops may contain loops themselves, corresponding to shapes inside the holes. In character recognition, it is frequently desirable to associate outline loops with those that contain them. A containing loop is termed the *parent*; the contained loop is the *child*.

Throughout this work, the containment of a loop shall be represented by the convention of labelling each loop with a level number. The outermost loops (not contained by any others) are at level 0. The loops immediately contained by level 0 loops (the first layer of holes) are at level 1. The level number increments with each successive containment such that a loop immediately contained by a loop at level n is at level $n + 1$. Furthermore, points or vectors on even level loops are ordered anti-clockwise and odd level loops clockwise so that the black pixel regions are always on the left of the line and white pixels always to the right.

The nesting of child loops in relation to their parents is represented through the ordering of the loops. All child loops appear in the ordering after their parents and before any higher level loops which are not their parents. For example, if loops A and B are level 0 loops and loops C and D are immediate children of A, and loops E and F are immediate children of C, then the order in which they are stored will be: A, C, E, F, D, B.

In this research, character data comes from a variety of sources and scanners. In each case the data has been converted to a run-length encoding for storage and as the input format to the vectorization process of chapter 3. It is also used as the initial format for outline conversion using Elliman and Connor's method [EC90]. The nested loop outline format is also used for storage at some stages and is the input format for the feature extraction process of chapter 5 and the approximate dictionary matching of chapter 7.

2.2.3. Types of Character Recognition

The field of off-line character recognition can be further divided into a variety of sub-fields devoted to the recognition of particular types of characters.

Early successes in the field were limited to the recognition of machine-printed documents. Machine print is generally easier to distinguish than human-drawn characters as the same characters will have approximately identical shape. Variations are caused only by scale (point size), font, typeface and noise. This is still a difficult task but early systems achieved useful results by limiting themselves to a single font and size. Modern systems recognize multiple fonts and sizes. They are accurate enough in practical use to save time but still fall short of 100% accuracy. The recognition output requires manual inspection to correct errors.

Another area where high levels of accuracy have been reported is in handprinted numeral recognition. Many experiments constrain the numerals in some way (e.g., boxes on a form [Güd76], spacing on a page) to allow them to be simply segmented. The recognition of the numerals is then a comparatively easy task as there are only ten classes to distinguish. Correct classification rates of between 85 and 99.79% have been reported but the differences in test data and constraints prohibit a meaningful, direct comparison. The 99.79% result reported by Masterson and Hirsch [MH62] used highly constrained data from trained writers and does not reflect the accuracy attainable in real applications. In practical applications, where noise is commonplace, the highest levels of accuracy claimed have not been achieved, even where there are contextual constraints on the digits.

More general handprinted character recognition extends to upper case letters, sometimes only the letters and sometimes mixed alphanumerics. A related area is the recognition of the 46 character Japanese Katakana set which has been the subject of much research effort in Japan. Suen *et al.* [SBM80] give a good review of both these fields.

Cursive script is the most difficult field in character recognition. A full cursive script recognition system must be able to discriminate upper and lower case letters as well as digits. The main obstacle is the fact that characters are joined. One approach to the problem is to segment the word into its component characters. This is particularly difficult and virtually impossible to achieve without any knowledge of what the characters are. The alternative is to attempt to recognize whole words rather than individual characters. This is only practical when the word can be constrained to a small lexicon.

Various other sub-fields of off-line recognition focus on particular applications or particular languages/alphabets. The recognition of postcodes (or ZIP codes) is one of the most popular applications (see Downton and Leedham [DL90b]). The main languages studied are English, Japanese, Chinese, Indian and Arabic [GS90] but many others have also been attempted, along with more unusual alphabets such as FORTRAN (e.g., Fujimoto *et al.* [FKH76]) and Pitman shorthand (Leedham and Downton [LD87]). Govindan and Shivaprasad [GS90] give a wide-ranging survey of applications and alphabets for which character recognition has been used.

This research is predominantly concerned with the recognition of modern English handwritten characters, i.e., Roman alphabet and Arabic numerals. Initial experiments on isolated characters avoid the problem of segmentation through the use of pre-segmented character databases (NIST and CEDAR, see section 2.6). Recognition of digits, and upper and lower case character sets has been investigated. Later experiments address the problem of segmenting handprinted words.

2.3. Preprocessing

The standard approach to character recognition that forms a part of nearly every fully-fledged system in the literature, is to preprocess the image representation in an attempt to make it easier to recognize. Whether such preprocessing is beneficial is discussed in chapter 4. For now let us consider the typical stages of preprocessing.

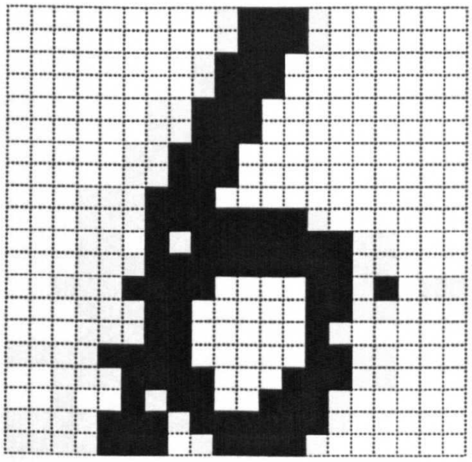
The desired functions of preprocessing can be divided into four main types: simplification of the character representation (by either thinning, vectorization, or line or curve fitting), filling or repairing of breaks in the image, smoothing of the representation to eliminate noise, and normalization to reduce differences between characters of the same class. Some, though not necessarily all, are used in the majority of off-line character recognition systems.

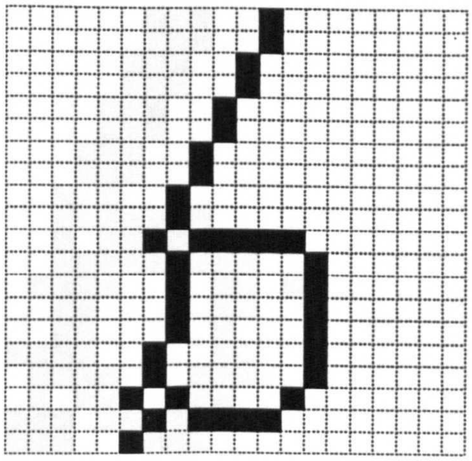
Figure 2.2 illustrates the types of preprocessing described in this section. Figure 2.2a shows the original image representation, in this case a bitmap of the letter 'b'. The image representations, b) to j), show the effect of different preprocessing operations on a). Note that the representations in b) to j) are not necessarily those that would be produced by any specific algorithm. They simply illustrate the typical results of the operations.

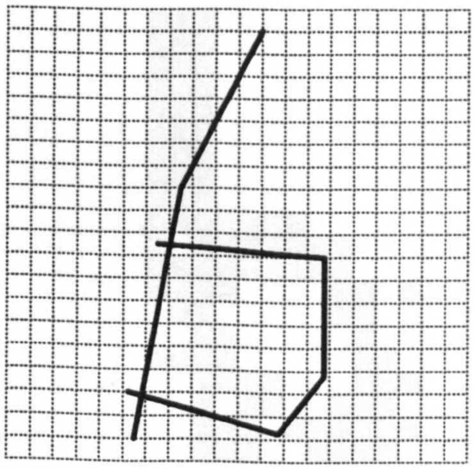
Many methods of performing preprocessing have been used. The most common technique is *masking*. This is a method applied to pixel-based representations and has been used not just for preprocessing but also for edge detection and feature extraction. A pixel mask (also called a template or window), usually 3x3 or 5x5 pixels in size, is moved over each pixel in the pixel map. A function, computed from the (usually) binary values of the pixels covered by the mask and fixed values associated with the positions in the mask, determines the new value associated with the pixel under the centre of the mask. In preprocessing and edge detection this new value will normally be a binary pixel value. In feature extraction the value may describe other information such as line slant or the presence of a topological feature.

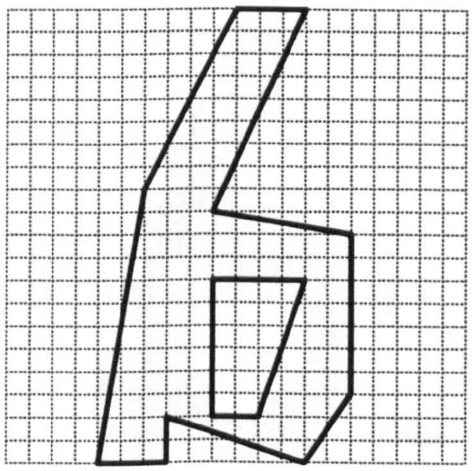
2.3.1. Thinning and Vectorization

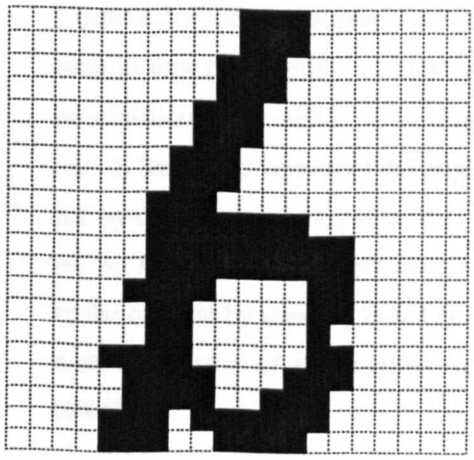
The most common attempt to transform the original image representation of characters into human abstractions of those characters is to change the representation of lines by areas of pixels (usually several pixels wide) to either unit-width pixel areas or zero-width lines. This

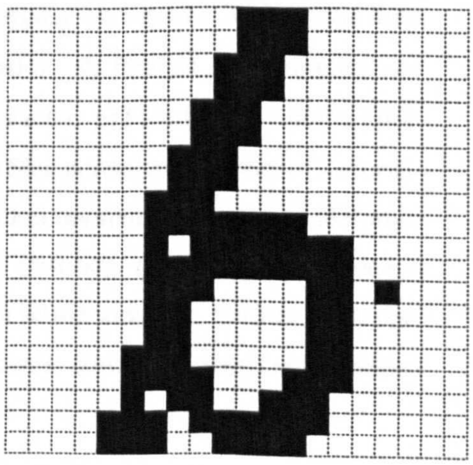

a) Original bitmap

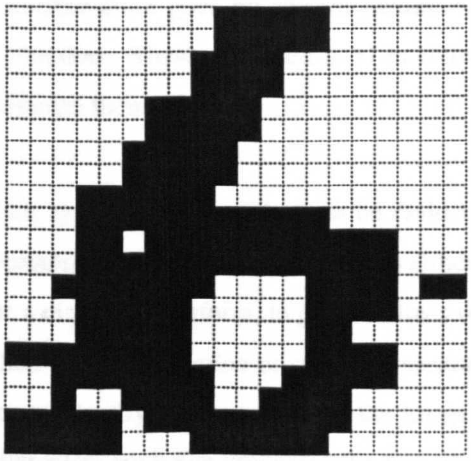

b) Thinned

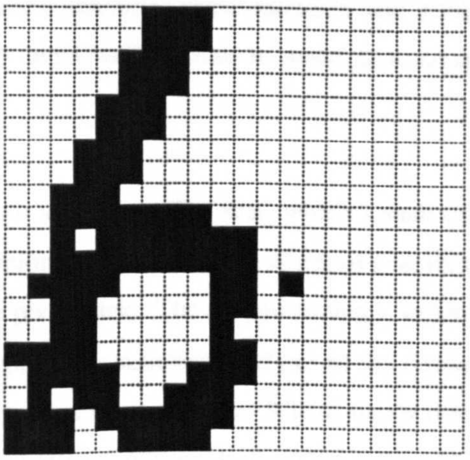

c) Vectorized

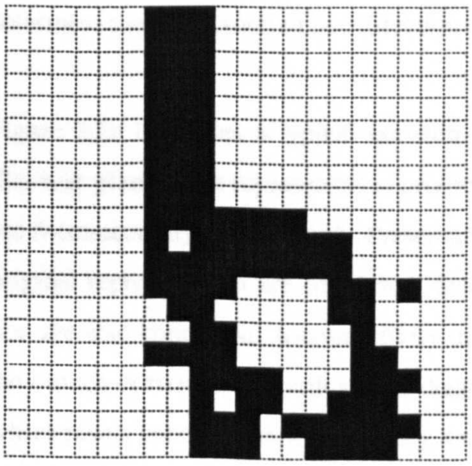

d) Polygonal approximation of outline


e) Filled


f) Smoothed


g) Size normalized to 20x20


h) Position normalized by displacement


i) Slant normalized by shearing

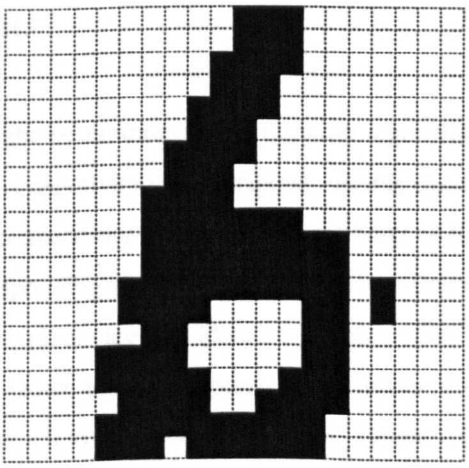

j) Line-width normalized by thickening

Figure 2.2 Examples of preprocessing. a) Original bitmap of a 'b'. b) Ideal thinning to unit pixel width. c) Ideal vectorization. d) Straight line polygonal approximation of the outline of the original character. e) Filled version of the original. f) Smoothed version of the original. g) Size normalized version of the original by Güdesen's method [Güd76]. h) Position normalized version of the original by Johnson *et al.*'s displacement method [JHL66]. i) Slant corrected version of the original by shearing based on the slant of the ascender. j) Line-width normalized version of the original by Güdesen's downward thickening method [Güd76].

transformation goes by several names: vectorization, thinning, skeletonization and stroke extraction. These are often used interchangeably but I shall choose to make distinctions between them.

This type of preprocessing can be divided into two strategies. The first, which I shall call *thinning* or *skeletonization*, involves either the erosion of pixels from the edges of characters or a distance transform, to produce lines only one pixel thick. The second strategy, which I shall call *vectorization* or *stroke extraction*, attempts some structural analysis of the character to identify line segments of the image and represent them as such. Where handwritten characters are being processed the line segments are often connected to form the paths of the pen strokes used in writing the character. The process may be extended further to attempt to determine the time ordering and direction of strokes.

Figure 2.2b illustrates a typical skeleton that might be produced by a thinning algorithm. The unit pixel width representation preserves the connectivity of the original. In this example 8-connectivity is preserved. Preservation of connectivity means that where parts of the original representation are connected, their corresponding pixels in the thinned representation are also connected. 8-connectivity means that pixels are considered to be connected if they touch horizontally, vertically or diagonally. Other algorithms preserve 4-connectivity, where pixels are only connected if they touch along a horizontal or vertical side. Figure 2.2c illustrates an ideal vectorization of the original bitmap. The figure shows the desired output from the vectorization approach but is not an actual output of any specific algorithm. Most algorithms would require some smoothing of the initial vectorization to produce such an ideal representation of the original shape.

The main aim of thinning is line-width normalization — producing the same skeletons from characters which differ only in the thickness of lines. This is a typical approach in structural character recognition as it aids in the detection of line crossings, endpoints and other structural features.

The aim of vectorization is in most cases to determine the path of the pen strokes making up the character, with a view to extracting higher level features based on the *way* a character is drawn, not just its final shape. This can be seen as an attempt to bring off-line character recognition closer to the field of on-line character recognition where the exact path of the pen is known. It is unrealistic, however, to expect to be able to accurately reconstruct pen strokes from an off-line image, particularly when strokes double back on themselves or otherwise overlap. The fields of on-line and off-line recognition remain distinct.

Thinned representations are often used as the first step towards determining the pen strokes. Once the character has been transformed to a one pixel thick representation it appears simpler to vectorize. The problem with the thinning first approach is that the local thinning operations are prone to errors caused by noise, such as spurs, and unwanted splits and joins. Distinguishing important features of the character from noise is not possible without the use of a wider context, e.g., the consideration of stroke width and line structure. The most difficult areas to vectorize are those where the pen strokes overlap, such as at junctions. Attempting a correct reconstruction of the strokes requires a structural examination of the lines approaching the overlap area. Thinning overlap areas to a single line first produces incorrect, distorted joins of the approaching lines which make subsequent correction difficult and in some cases impossible. Pure vectorization methods have been developed in an attempt to tolerate noise and correctly vectorize overlap areas, particularly junctions.

Thinned representation formats are often the same as the original format (bitmaps, run-length, etc.). Vectorized representations require different file formats, usually storing either a list of coordinate points or a list of vectors. Since the aim of thinning and vectorization is to obtain a line-based representation of characters, preferably stroke-based, the formats are often translated immediately to an alternative representation. This can be seen as a classification of skeletons or vectors into higher level primitives. These primitives may be represented in terms of stroke ends (position and connection of endpoints, junctions, corners, etc.) or in terms of strokes (position, length, curvature, orientation of strokes, and their connections to other strokes). Higher level primitives offer the possibility of higher level features which may have more powerful discriminatory ability. The disadvantages of translating to stroke primitives are that stroke-based or stroke-end-based formats may lose precise shape information and that the chosen primitives may be incorrect. This form of representation is closely related to geometric and topological feature extraction (see section 2.5.4).

The literature on thinning and vectorization is huge. This section aims to give an overview of the important work in the field. More extensive surveys of thinning and vectorization methods can be found in Tamura [Tam78], Davis and Plummer [DP81], Hilditch [Hil83], Naccache and Shinghal [NS84], Smith [Smi87] and most recently Lam *et al.* [LLS92].

2.3.1.1. Pixel Erosion Methods

The earliest method used for thinning was the pixel erosion method, or iterative method, used on pixel-based image representations. Layers of pixels are successively removed from the boundary until only a skeleton remains. The criterion for removing or keeping a black boundary pixel, p, is based on its eight neighbouring pixels and is computed using a mask centred on p. These masks are usually 3x3 but larger 4x4 (e.g., Holt *et al.* [HSC87]), 5x5 (e.g., Favre and Keller [FK83]), kxk masks ($k \geq 3$) (O'Gorman [OGo90]) and non-square masks (Sukuki and Abe [SA87]) have been used. The early algorithms used averaging operations or matching against specific mask patterns to determine the retention or deletion of p. The approach has seen much development since its first use by Dinneen in 1955 [Din55].

A fundamental problem with the early algorithms was that the local analysis of pixel windows did not account for connectivity in the image and consequently connected parts of the image would become disconnected during the thinning process. Most differences in thinning algorithms are caused by their differing methods of dealing with this problem and whether they consider black pixels to be 4-connected or 8-connected. Three criteria for determining connectedness have been used: crossing number, connectivity number and pixel simplicity. The value of these measures can be used to determine whether removal of p would alter the connectivity of the skeleton.

The crossing number was first proposed by Rutovitz [Rut66] as the number of transitions from white to black or vice versa when the eight neighbours of p are traversed anticlockwise. Hilditch [Hil69] proposed an alternative crossing number using a variant traversal of the neighbouring pixels which cuts the corners between black neighbours which are horizontally connected to p.

Yokoi *et al.* [YTF73] proposed 8-connectivity and 4-connectivity numbers which are more easily computable than the crossing numbers. The 8-connectivity number equals the number of times *p* would be traversed in a contour-following algorithm for a connected component. Pixels to be kept are those that are traversed twice and are called "multiple" pixels.

Deletable pixels are sometimes called "simple". Simple pixels have been defined based on whether their removal would alter the connectivity of the whole image. Equivalently, Tsao and Fu [TF81] defined them in terms of the genus of the image — the number of connected components in the image minus the number of holes. The effect of removal of *p* on the genus of the image can be determined solely from the eight neighbours of *p*, usually using a look-up table. If the genus remains the same then *p* is simple. Rosenfeld and Davis [RD76] implemented a parallel genus-preserving algorithm.

Further differences in thinning methods are due to the order and scope of their examination of pixels. A raster scan (left to right, top to bottom) is commonly used, where all pixels in the image are tested for deletion. Naccache and Shinghal's safe-point-thinning algorithm (SPTA) [NS84] used two raster scans per cycle, one horizontally (to find west and east points for retention) and one vertically (to find north and south points for retention). A popular alternative method is contour following, where only the border pixels of each component are examined in each iteration. Contour following algorithms are faster as they visit fewer pixels.

Another cause of differences in thinning algorithms is the mode of operation: either sequential or parallel. Sequential algorithms examine each pixel in a predetermined order in each iteration. Each deletion within an iteration affects all subsequent examinations in that iteration. In a parallel algorithm each pixel is examined independently based on the state of the bitmap at the start of the iteration, thus deletions within an iteration have no effect until the following one. Independent examination of pixels can be achieved in a sequential algorithm by marking pixels for deletion rather than removing them immediately. At the end of each iteration a second pass is made to remove all the marked pixels. However, there are advantages to non-independent sequential deletions. Marking or parallel methods can remove large areas in a single iteration and have trouble preserving connectivity as they do not generally take into account the effect of removing pixels adjacent to marked ones. Two-pixel-wide branches could be entirely deleted in a single iteration of these methods, whereas a non-independent method would retain one-pixel-wide skeletons of the branches. To prevent this occurrence many sequential methods use the condition, first proposed in a fundamental thinning algorithm by Hilditch [Hil69], that at least one neighbour of p must be unmarked, otherwise p cannot be removed.

Many sequential thinning algorithms have been used. Hilditch's algorithm [Hil69] used her crossing number to preserve connectivity and two-pixel-wide lines. Many researchers have implemented, extended and modified the Hilditch algorithm. For example, Riazanoff *et al.* [RCC90] included a convexity measure in the deletion criteria and O'Gorman [OGo90] extended Hilditch's criteria to *k*x*k* pixel masks.

Six 3x3 masks which detect break points (points whose removal would affect connectivity) were used by Beun, Chu and Suen, and Pavlidis. Beun's original method [Beu73] identified edge pixels and deleted them if they did not match any of the break point masks. Prior smoothing was required to eliminate the many spurs, caused by incorrect edge pixel identifications, that would otherwise have occurred using this algorithm. Chu and Suen's algorithm [CS86] incorporated a smoothing cycle at each iteration which moved skeletal pixels closer to one of their horizontal or vertical neighbours if the neighbour was further from the background. This produced smoother skeletons whose pixels were closer to the medial line of the image. A problem with these methods was that they could allow excessive erosion and shortened branches. Pavlidis [Pav82a] solved the problem by combining a four-subcycle parallel examination of the pixels with the method of marking pixels for deletion as used in sequential algorithms (and not deleting them until the end of the *iteration*). Skeletal pixels were also marked and retained throughout the process.

Several algorithms use the Rutovitz crossing number. Arcelli and Sanniti di Baja's deletion conditions [AS78] using this number preserve connectivity but can cause reduced corners and incorrect end points. Additional criteria were suggested using a second crossing number: the number of transitions between contour and noncontour pixels, where contour pixels are defined as black pixels with at least one white neighbour among their surrounding eight pixels. They developed their method further to preserve significant features of the image [AS80] [AS81]. Protrusions were identified in the image, whose distance from the interior pixels exceeded a threshold. These protrusions were retained while thinning was performed as in [AS78] and then thinned to one or two pixels thick by use of a labeling function. Pixels satisfying a maxima condition under the function were retained while the rest were deleted.

Pavlidis [Pav80] proposed a sequential method based on retention of "multiple" pixels. A multiple pixel is defined by Pavlidis as one which is traversed more than once in a contour tracing or has no neighbours in the interior of the object or has at least one neighbour on the same contour that does not come immediately before or after it in a contour tracing. Since the multiple pixels alone do not necessarily preserve connectivity, Pavlidis constructed skeletons by defining skeletal pixels. Skeletal pixels are either multiple pixels or pixels with a neighbouring skeletal pixel from a previous iteration. The method has been implemented using a masking technique [Pav82a]. Ferreira and Ubéda [FU94] developed a fast parallel contour tracking algorithm which they used to implement this method. The algorithm has been shown to preserve connectivity, but it produces skeletons wider than unit pixel width. Arcelli [Arc81] proposed an alternative algorithm, based on stripping non-multiple pixels from the contour, to reduce the thickness of the skeletons. As in [AS80] and [AS81], the algorithm attempted to preserve significant features (protrusions) and eliminate noise by retaining some non-multiple pixels and removing some multiple ones. This was done via analysis of the chain codes of the components.

An alternative to stripping pixels from contours is to generate successive new contours inside the current one until no more can be generated and a skeleton is left. With this approach it is much simpler to preserve two-pixel-wide lines and check connectivity. The approach was introduced by Xia [Xia86] who generated contours by sequentially reducing runs of three contour pixels to a single pixel — the one closest to the bisector of the angle formed by the three pixels. Xu and Wang [XW87] generated contour edges in the north, south, east and west directions individually, using 3x3 masks on edge pixels in each direction. Significant development of the method was made by Kwok [Kwo88] who used the identification of safe points (as in Naccache and Shinghal's SPTA method [NS84]) in combination with chain codes to determine the new contours, and produced the chain codes for the new contours at the same time. Kwok compared his algorithm favourably with [Pav80], [NS84], [ZS84] and [XW87].

Parallel algorithms prevent excessive pixel erosion by using multiple subcycles or subiterations within each iteration. Commonly four subcycles are used, each subcycle deleting edge pixels in one of four directions (north, east, south and west), e.g., Stefanelli and Rosenfeld [SR71]. Deletions in each subcycle affect the subsequent subcycles within the iteration. The directions have been combined to require only two iterations, e.g., north and east in one and south and west in the other. Stefanelli and Rosenfeld [SR71] also used such a method. They prevented the excessive deletion of edge pixels by storing all skeletal pixels (pixels that are on one-pixel-wide lines) at the start of each subcycle and replacing them after the edge pixels have been deleted. Another technique for preventing excessive erosion, called border parallel operation, is to only delete pixels if they meet the deletion criteria in relation to their neighbours, both as they are in the current subcycle and as they were at the start of the iteration (Hilditch [Hil83]).

The seminal algorithm for parallel thinning was conceived by Rutovitz [Rut66]. This one-subcycle algorithm used a 4x4 mask and Rutovitz's crossing number to preserve connectivity. The algorithm has seen many developments and modifications. Deutsch [Deu69] modified the algorithm to thin two-pixel-wide diagonal lines to one-pixel-wide, which does not occur in the original method. Deutsch also rotated some of the deletion criteria to centralize the skeletons and reduce the necessary mask size to 3x3 [Deu72]. Zhang and Suen's two-subcycle version of Deutsch's algorithm used a much simplified subset of the deletion criteria [ZS84]. Lü and Wang [LW86] modified Zhang and Suen's version to retain two-pixel-wide diagonal lines; this was later extended by Wang *et al.* [WHF86] to thin those areas to unit pixel width. The modified two-subcycle methods, which prevent the excessive pixel erosion of the original algorithm, were reduced to one subcycle by Holt *et al.* [HSC87] by returning to the larger context of a 4x4 mask to spot the vulnerable two-pixel-wide lines.

Recent work on parallel thinning has concentrated on one-subcycle algorithms on the assumption that they will be faster than two-subcycle algorithms. Lam *et al.* [LLS92] suggest that this is not necessarily the case as the one-subcycle algorithms require analysis of larger pixel windows to preserve connectivity and hence each subcycle is slower than the two-subcycle methods (which generally use 3x3 pixel windows). Nonetheless, several single subcycle algorithms have been proposed.

Favre and Keller [FK83] used a 5x5 window to label the pixels in the usual 3x3 window as either core, interior, rim or skeleton, and applied deletion criteria based on these labelings. Chin *et al.*'s one-subcycle algorithm [CWS87] used sixteen 3x3 templates (to detect borders, corners and spurs) and two 4x1 or 1x4 templates (to detect two-pixel-wide lines). Pixels matching the 3x3 templates were deleted but only if they did not match the 4x1 or 1x4 templates. Chen and Hsu [CH89] used a 5x5 window with deletion criteria using the Hilditch crossing number and tests for p being in a two-pixel-wide diagonal line and for the local connectivity of p's neighbours.

Many other pixel erosion methods have been used, varying in their connectivity measures, in whether they are raster scanning or contour following, and in whether they are sequential or parallel. These methods operate quickly, with computation time dependent on the maximum width of the component. However, the potential increase in speed from parallel implementations is limited by the requirement for larger context and multiple subcycles to preserve connectivity information.

The problem with pixel erosion methods, in general, is that they are sensitive to noise on the edges of components and often produce spurious tails on the skeletons. Their local examination is insufficient to distinguish spurs that are caused by noise from spurs that are features. Another problem is that the skeletons produced are not invariant under rotation because of the order in which pixels are removed, i.e., the order of examination in sequential algorithms or the order of subcycles in parallel algorithms.

On clean images, local operations can preserve the connectivity and topology of an image, but in many cases they do not preserve the geometry of characters [LLS92]. The accuracy of skeletal representation is particularly poor at junctions and other areas of pen stroke overlap, which require a consideration of the wider context of incoming pixel lines. Pixel erosion techniques do not aim to produce an approximation of the path of the pen strokes used to draw the character; they simply aim to find a medial line for the image. Their original use was on machine-printed characters, where the concept of pen stroke has limited relevance, but for handwritten characters the methods need to cater for pen strokes which cross, overlap and double back on themselves. Simply thinning the characters is insufficient and a more global examination is necessary.

2.3.1.2. Distance Transforms

Another approach to thinning is to apply a transform to the image representation which converts connected components to a skeletonized form based on the distances to their boundaries. This makes use of slightly more global information than pixel erosion methods. Transforms of this kind go by several names: *medial axis transform* (MAT), *symmetric axis transform* (SAT) and *grassfire* (or *prairie fire*) transform.

The medial axis transform is commonly based on a variety of discrete distance transforms (DTs), first used for this purpose by Blum [Blu67]. Each point in a discrete grid is assigned a distance value, indicating the distance to the boundary of the object containing it. This is often thought of as the radius of the largest circle that can be drawn, centred on the point, within the confines of the boundary. One definition of the MAT is that if a point, x, has more than one boundary point at the closest distance, $r(x)$, then x lies on the medial axis. The set of all points on the medial axis forms the skeleton. By retaining the value of the closest distance it is possible to reconstruct the original image from the medial axis. For each point x on the medial axis, a disc, radius $r(x)$, is centred on x. The union of the discs gives the original shape.

The distance map can be calculated using a variety of approximations to the true Euclidean distance, varying in accuracy and ease of computation. Rosenfeld and Pfaltz [RP68] and Arcelli and Sanniti di Baja [AS89] used the *city block* DT where distance between two points is measured as the number of 4-connected (horizontal or vertical) steps between squares on the grid, along the minimal path between the points. Arcelli and Sanniti di Baja [AS85] also used the *chessboard* DT where distance is measured similarly to the city block DT but steps between grid squares are 8-connected (horizontal, vertical or diagonal). A *hexagonal* DT on a hexagonal grid was used by Meyer [Mey88], and Borgefors and Sanniti di Baja [BS88]. Borgefors [Bor84] used a *chamfer* DT, where the difference in length between diagonal steps and horizontal and vertical steps is simply approximated, e.g., horizontal and vertical steps are assigned the distance value 3 and diagonal steps are assigned the distance value 4; other *quasi-Euclidean* DTs were used by Montanari [Mon68] and Dorst [Dor86]. The true *Euclidean* DT was used by Danielsson [Dan80], Yamada [Yam84], Ho and Dyer [HD86], and Klein and Kubler [KK87].

The skeleton is extracted from the distance map using a range of techniques. Montanari [Mon68] observed that each point in the map has a minimal path to the boundary and medial axis points are those that are not on the minimal path from any other point. The problem of skeleton extraction was therefore converted to one of finding a minimal cost path through a graph. A more commonly used approach is Blum's original method [Blu67] which uses the analogy of a grassfire. Each point in the distance map corresponds to a blade of grass. The boundary of the object is the source of a fire, producing wavefronts (or fire fronts) which propagate across the distance map according to some rule. The skeleton is defined as the locus of meeting wavefronts (called quench points). Another common method is ridge following, where maximum and saddle points are found on the surface of the map. An example of this was presented by Dorst [Dor86].

The city block and chessboard DTs based on connectivity are simple to compute and can easily preserve the connectedness of the object. However, they are variant under rotation and are not as accurate as the quasi-Euclidean and Euclidean DTs. Ridge following extraction must be used on hexagonal, quasi-Euclidean and Euclidean DTs. The skeletons are more accurate and smoother but it is harder to preserve connectivity. Broken skeletons must be rejoined using growing procedures along the maximum gradient directions on the map [Dan80] [Dor86] [HD86] [KK87].

The advantages of distance transforms over iterative thinning methods are that they contain symmetry information, are generally faster, and the true Euclidean DTs produce invariant skeleton topology under rotation, translation and scaling. However, distance transforms have several flaws. Medial axis transforms are extremely sensitive to noise; even a single pixel can cause large changes in the shape and structure of the medial axis. Commonly, small amounts of noise on the contour cause many spurs, making the skeletons unsuitable for character recognition where the aim is not to preserve every small anomaly but to extract the essential shape of like characters in a simple and similar form. An approach to overcoming this problem, called the multi-resolution symmetric axis transform, was proposed by Pizer *et al.* [POB87]. This involves producing a hierarchical tree structure of medial axis components at different scales corresponding to the conventional MAT for different degrees of outline smoothing. While this approach gives a complete and robust shape description at each scale, there remain problems in producing similar multi-resolution SATs for similar shapes [Mar89].

Further problems with distance transforms are that they do not correctly represent the shape of concave portions of the boundary and do not produce correct skeletonizations of junctions (since they pay no attention to stroke directions) or endpoints (which generally appear as Y-shaped skeletons). Nonetheless, the relative simplicity of their definition and implementation, and the speed of their operation makes these methods popular with some researchers.

2.3.1.3. Vectorization Methods

To overcome the problems of spurs and the incorrect treatment of overlapping pen strokes, particularly at junctions where an accurate reconstruction should be possible, an alternative to the pixel erosion and distance transform approaches has emerged. This approach, generally referred to as vectorization, considers a larger area than those usually examined by mask-based techniques and hence can use a wider context in determining the correct path of strokes in a character.

A very simple vectorization method is to construct a skeleton by joining the midpoints of adjacent, overlapping black run-lengths. In its basic form, this method considers little global information but it is the forerunner of a more useful vectorization algorithm. The basic method itself is very fast but is unsuitable for character recognition as it only works effectively when strokes run roughly perpendicular to the scan lines.

The problem of vectorizing characters with this method was addressed in a number of papers by Pavlidis. The run-length representations were represented by a line adjacency graph (LAG) [Pav78] where each black run-length is a node. Arcs between nodes indicate that the associated run-lengths are on adjacent scan lines and overlap (in other words they indicate that the run-lengths are touching). In one algorithm [Pav82c], he performed sequential pixel erosion first and then constructed the LAG, which he then vectorized by line and curve fitting. This thinning first approach suffers the noise and overlap problems of the initial iterative thinning; however, analysis of the LAG provided a wider context which allowed some of the noise problems to be corrected.

To limit the problems caused by the sequential pixel erosion, Pavlidis identified groups of similar length black pixel runs that were adjacent, overlapping and formed lines in a direction approximately perpendicular to the scan lines [Pav84]. These could be simply vectorized by the basic run-length method mentioned above. The remaining areas were thinned by a sequential pixel erosion.

In 1986 Pavlidis avoided pixel erosion thinning completely using a variant of the LAG [Pav86]. Sets of adjacent, overlapping black run-lengths of approximately the same length are grouped to form a single node in his compressed line adjacency graph (c-LAG). The arcs of the c-LAG indicate the connections between groups. The c-LAG is created by transforming the original LAG to a graph that is homeomorphic to it but has the minimum number of nodes. Groups with only one overlapping group, either above or below, are called *path nodes* and are vectorized as either a horizontal line or a line joining the midpoints of the top and bottom run-lengths in the group, according to a height to width comparison. Groups with more than one overlapping group above or below are called *junction nodes*. The path of vectors across the junctions is determined by "compound vectorization" rules, based on the angles at which the possible vectors would cross the junction. This algorithm has been developed in this thesis and is discussed in more detail in section 3.2.

Pavlidis's 1986 algorithm is fast because it works from run-lengths but is still hampered by the one-directional treatment of characters. Much of the complexity of the algorithm could be avoided through the consideration of vertical pixel runs as well as horizontal run-lengths. This was the basis for the new vectorization method of chapter 3. Another approach to this problem was proposed by Sinha [Sin84] [Sin87a] who labeled boundary pixels as either horizontal, vertical, left-going diagonal, right-going diagonal or "don't care", based on a local analysis of the boundary. Pixels with the same label on opposite sides of the boundary indicate a matched pair. The midpoints of these pairs indicate potential skeleton points. Primitives representing lines in specific directions are searched for by finding pairs with the same label occurring in roughly distinct groups in the image. Where strokes vary in direction, and particularly where they do not run close to the horizontal, vertical or 45° diagonals, the method has trouble detecting primitives as the labels vary greatly within groups of neighbouring pairs. The method is not considered sufficiently robust for general applications [LLS92].

Another approach to vectorization is contour tracking. Opposite edges of the image are tracked simultaneously; the midpoint of the two edge trackers is added to the vector at each stage. Dessimoz [Des80] used this method, keeping minimal distance between the edge trackers to keep them synchronized as they move round the outlines. This minimal distance method was prone to errors where line thickness varied suddenly and where close lines ran parallel. Determining a tracking criterion which gives accurate vectorization in all circumstances has proved very difficult.

By treating the image representation as a many-sided polygon, Shapiro *et al.* [SPS81] tracked contours with a series of trapezoids, again using minimum distance as the criterion for keeping the trackers synchronized around bends. The midpoints of the trapezoid sides indicated points on the vectors. This method was applied to ribbon-like objects, such as DNA molecules, which consist of a backbone object with small branches along the length. The method was not generally applicable and was unsuitable for character recognition as it behaved poorly on junctions and did not allow for further sub-branches from the branches. Baruch [Bar88] used a similarly limited algorithm, using rectangular windows of variable dimensions to follow elongated objects and vectorizing them by joining the centres of the rectangles.

Martínez-Pérez *et al.* [MJN87] used a contour tracking method that stepped around the vertices of the image representation. A midpoint was only added to the vector chain when certain projecting lines from the vertex intersected the opposite line segment within a threshold distance. For convex contour points the projecting line is the bisector of the interior angle at the point; for concave contour points, two projecting lines are tested, the normal lines of the two line segments forming the vertex (called *pseudonormals*).

Contour tracking methods are prone to errors caused by the problem of keeping trackers approximately synchronized as they move round the contour. They also require some thresholding to prevent incorrectly creating vector points in the middle of thick areas. Martínez-Pérez *et al.*'s algorithm is rather more complicated and computationally expensive than the run-length-based methods where pairing of edge points is simplistically but uniformly achieved and distance thresholding is implicitly performed in the identification of horizontal or vertical line segments.

Contour tracking methods have not commonly been applied to the vectorization of overlapping pen stroke areas, though they are well suited to the handling of junctions. The presence of junctions could be simply indicated where the distance between trackers exceeds a threshold. Accurate vectorizations of these junctions could be determined based on the vectors of the surrounding non-junction areas (lines). This approach can similarly be applied to run-length-based methods (as has been done in chapter 3).

Priestnall's contour tracking method [Pri94] addresses these issues. The line segments of a smoothed contour are tracked and pair segments are found on the opposite side of the contour by matching the lines with certain criteria: degree of overlap, average distance and angle between the lines. This is more computationally expensive than the previous tracking methods but takes more care in keeping the edge pairs synchronized and limiting their separation. The pairs form trapezoids which are vectorized along their medial lines. The unpaired areas are classified as either junctions, endpoints or corners and this information is used to guide their vectorization, which is based on the incoming vectors from the adjoining paired areas. This gives much better vectorization of stroke crossings than the previous methods.

For character recognition the vectorization approach appears generally superior to the pixel erosion and distance transform approaches. It is more tolerant to noise and allows for much improved vectorization of junctions. It should be remembered, however, that the evaluation of correctness of skeletonized forms is largely a subjective one and varies considerably between researchers and applications. Research continues on each of the thinning/vectorization approaches.

2.3.2. Polygonal Approximation

Polygonal approximation, also called piecewise approximation and line or curve fitting, is an encoding of outlines as a much smaller number of line segments that preserves the original shape or minimizes the error between the data points and the fitted lines. The line segments are usually straight but higher-order approximations may also be used. An example of the representation which might be produced by a straight line polygonal approximation is shown in figure 2.2d.

Polygonal approximation is in one sense a preprocessing operation that simplifies and smooths character outlines to aid subsequent geometric and topological feature extraction. However it may also be viewed as a feature extraction process itself, with each of the approximated line segments forming features, e.g., position, angle and length of line. These features are usually concatenated into a string and classified using a string-based template matching technique (see sections 2.5.4 and 6.3.2). Polygonal approximation is also used as a preprocessing stage for outline-based thinning and vectorization methods.

Most techniques for polygonal approximation perform interval splitting or merging, or a combination of the two. Interval merging involves splitting the outline into many small sections, e.g., straight lines, and iteratively merging adjacent sections to create larger sections which still fit the outline. Splitting starts with a single section approximation of the outline. If the approximation does not fit sufficiently well, the poorly approximated portion of the outline (in the initial case, the whole outline) is split in two and approximations are made to the two new portions. The process is repeated until the whole outline has been sufficiently approximated. Ramer [Ram72] used a splitting method for straight line approximation that split the outline portion at the furthest point from the approximating line.

Neither splitting nor merging are sufficient on their own. Splitting can produce adjacent approximations which could be merged within the parameters of the fit. Merging leads to poorly placed interval boundaries. Where error is tolerated by the fitting criterion the merged regions can extend round corners before the error limit is reached. A combination of splitting and merging is required for accurate polygonal approximation.

Pavlidis [Pav77] gives a comprehensive discussion of polygonal approximation techniques up to 1977, including his own split-and-merge technique [Pav76]. Many new methods have been published since then to improve the accuracy of approximations and to preserve important points such as corners. Liao [Lia81] applied a second-pass to Ramer's method to fit conic arcs in addition to straight lines. Pavlidis [Pav82b] formulated curve fitting as a pattern recognition problem and used features, such as ratio of curve length over endpoint distance and number of sign changes of the error function, to improve the accuracy of approximation acceptance. Abe *et al.* [AAH91] detect dominant points to aid the approximation. To optimize the choice of approximations, strategies such as minimizing the perimeter formed by the approximation (Sklansky *et al.* [Skl70] [SCH72]) and matching the area of the approximation as closely as possible to the original outline area (Wu *et al.* [WDR81]) have been used.

Many more techniques for polygonal approximation have been published but few are directly concerned with character recognition — the reason being that there is little similarity between approximations of similar characters so approximation strings are unlikely to make good features (see section 2.5.4).

2.3.3. Filling and Joining

Filling is the elimination of small white pixel regions in black pixel areas, and vice versa. These small areas are probably noise. An example of a filled image is shown in figure 2.2e. In this example the original bitmap of figure 2.2a is considered to be 4-connected for the purpose of filling. Any white pixels whose 4-neighbours are all black have been changed to black, and any black pixels whose 4-neighbours are all white have been changed to white. (The 4-neighbours of a pixel are those pixels which are 4-connected to it.)

Filling is almost always performed using a masking technique. Various sizes of masks have been used; 3x3 pixels is the most common size, e.g., Güdesen's 24 filling masks [Güd76], but masks as large as 8x6 pixels have been used to fill larger spots of noise [SM83]. The operation is often regarded as a form of smoothing. The two are closely related; masks for filling may easily be combined with masks for smoothing so the two preprocessing stages can be performed in essentially the same operation.

Another related technique is the repairing of breaks in the image by joining the components on either side. These breaks are generally much larger than the small pixel regions removed by mask-based filling. Systems using feature extraction methods which can only operate on one connected component at a time often use this technique to merge multi-part and broken images into a single connected component.

Banks [Ban91] identified components to be joined by finding their endpoints from a vectorized representation and measuring the distance from a component's endpoints to points on the other components. If this distance fell below a certain threshold, with the additional criterion that there was some vertical overlap of the components, then the components were joined with a line extending from the endpoint to the other component, in a direction perpendicular to the closest line on the other component.

This method is reasonably intuitive for vectorized images but the joining methods for outlines and bitmaps are less obvious. Although many researchers have used joining to repair breaks in images, the method is difficult to generalize into a solid theory and consequently little has been published on this form of preprocessing.

2.3.4. Smoothing

The appearance of many character representations is unsatisfying to human eyes. Pixel-based representations may have unsightly bumps around their edges. Outlines may have jagged, step-like orthogonal edges and bumps of noise. Vectors may have spurs or may have too sharp angles on lines which should be smoothly curved. These may be caused by noise in the original image or may be products of the sampling characteristics. One of the most common forms of preprocessing, therefore, is to "smooth" the representation to a more aesthetically pleasing form.

Aesthetics are not the only aim however. The reasoning behind smoothing of outlines and vectors is that it reduces the data required to store the characters and produces a simpler representation, hopefully preserving the original shape but without the small distortions and noise. This should reduce the differences between characters of the same class. Removing small distortions from the representation can be seen as an attempt at style normalization.

Figure 2.2f shows an ideal smoothing of a bitmap. Note that most smoothing techniques would alter the same pixels as filling would (see section 2.3.3). These pixels have deliberately been left unchanged in the figure to highlight the different aims of smoothing.

Smoothing of pixel-based representations is performed using mask methods. It is closely related to pixel erosion thinning methods: thinning in this manner can be seen as heavy, repetitive smoothing. As with pixel erosion methods, mask-based thinning involves changing the value of pixels according to the values of their surrounding eight neighbours. This may be by comparison to particular template patterns or by more general rules. Many variants exist. For example, Stentiford and Mortimer [SM83] smoothed spurs from bitmaps by eliminating all black pixels with less than three black neighbours and with Yokoi *et al.*'s 8-connectivity number (see section 2.3.1.1) equal to one. Bozinovic and Srihari [BS89] regularized contour smoothness by making a pixel black if and only if the number of neighbouring black pixels exceeded a threshold.

Outline smoothing via polygonal approximation has been discussed above (section 2.3.2). A simpler type of outline smoothing, which can also be applied to vectorized representations, is performed by examining sequences of three points on the outline or vector. The middle point is removed if its perpendicular distance to the line joining the other two points falls within a smoothing threshold. This suffers from similar problems to polygonal approximation; important points such as corners can be removed when other points are close to them. This can be countered by detecting and preserving points at sharp angles. Another problem with this approach is that it will typically remove too many points from smooth curves.

A variant method of outline smoothing, by Ho and Dyer [HD86], works not on the boundary but on the medial axis transform (see section 2.3.1.2). The MAT is pruned based on the *relative prominence* of its associated boundary segment. Each point on the MAT is associated with its two closest points on the boundary. The relative prominence is a measure of the maximum deviation of the boundary segment between these two points from the circular arc (centred on the MAT point) joining these two points. An elimination process, based on thresholding the relative prominence of all MAT points, gives a scale-independent means of distinguishing noise from important features. After noise is removed, a smoothed outline can be recovered from the MAT.

2.3.5. Normalization

Normalization is commonly applied to reduce certain forms of variation between like characters. This reduces the sizes of the training sets required to learn character classes. Some forms of normalization do not alter the essential shape of characters while others can produce significant differences from the original shape.

The most common of these preprocessing types is *size normalization*. Characters are scaled or otherwise converted to a normalized width or height (or both). In systems where size invariance is not achieved by the method of feature extraction or by the classification method, this normalization is a useful and popular technique for providing such invariance.

Strategies for size normalization vary only slightly. Characters are either elongated from a point of origin or are fitted to a fixed size bounding box. Some methods scale the height and width by an equal factor. The factors for both directions are found and the smaller one is used to scale both dimensions. Other methods scale the height and width independently to fit the bounding box. The former method preserves the aspect ratio of the original shape, whereas the latter method can introduce large distortions in cases where the height and width are very different.

Size normalization of outlines and skeletons is simple and intuitive if the coordinates are allowed to be non-integer. Where integer coordinates must be preserved, such as for bitmap representations, the normalization is more complicated as one has to take rounding effects into account.

Hussain *et al.* [HTD72] performed size normalization on character bitmaps, provided their height-width ratio exceeded a threshold. This was to prevent tall thin characters such as '1' being expanded to fill the whole bounding box. Güdesen [Güd76] used a size normalization technique which scaled bitmaps to a normal window size. The height and width of a character are scaled independently to fit the window. The scaling of a row of a character with width w to a normalized width, W, is performed by first expanding the row to a binary vector of size $w \times W$. The w pixels in the character translate to w consecutive runs of W pixels in the binary vector; the colour of each run matches the corresponding pixel in the character. The expanded vector is then sampled in W blocks of w pixels, each block corresponding to a position in the normalized row. The number of black pixels in each block is counted. The pixel in the normalized row is set to 1 if the count exceeds a threshold value and to 0 otherwise. This conversion is applied to each row, and then each column is scaled to a normalized height by the same method. Figure 2.2g shows the original bitmap of figure 2.2a size normalized to a 20x20 pixel bounding box by Güdesen's method.

Güdesen reduced the error rate in his handprinted character recognition experiments by a factor of six using this technique. His analysis of preprocessing techniques concludes that normalization of character dimensions is by far the most effective method of improving recognition accuracy.

A common problem with feature extraction methods is that measurements from similar shapes sometimes vary because different starting points or reference points have been used by the extraction process. Some types of feature are invariant to the position of the character; others allow for simple relocation of the reference point. Many however, particularly those that operate on pixel-based representations, are reliant on characters falling in analogous positions in the original presentation of the input frame. These methods can benefit from a preprocessing stage, called *position normalization*, which aims to shift the original characters so that they are always presented to the feature extractor or classifier in analogous positions.

There are essentially two strategies for this. One is to centre the character according to its centre of mass or the centre of its dimensions. The other is to displace the character so that its sides align with a reference frame, e.g., the character is moved towards a corner of the image field so that it touches both of the two limiting lines of the right angle.

Johnson *et al.* [JHL66] displaced characters by left-justifying them. An example of position normalization by Johnson *et al.*'s displacement method is shown in figure 2.2h. Güdesen [Güd76] experimented with the technique and found that classification results were best when characters were displaced towards the upper right-hand corner. He also applied a centering method to the same data. Characters were centred on the bitmap input frame according to their centre of gravity (determined by analysis of their moments). He found this to be slightly better than displacing, though the centering method allowed parts of the character to be shifted out of the input frame in some cases. Other examples of centering methods are Fu *et al.* [FCC67], who centred characters within a circular reference frame, and Highleyman [Hig62].

Slant correction (or *de-skewing*) attempts to normalize characters to a uniform obliquity. Most writers produce some natural slant in their writing; the range of possible slants makes the task of recognizing characters more difficult. Segmentation of characters by vertical lines is also particularly difficult when writing is slanted. To simplify these tasks, slant normalization tries to adjust all characters to a uniform angle of inclination, normally vertical. There are two ways to do this: by rotating the character or by shearing it.

Rotating methods must first determine the angle of inclination of the original figure. Banks [Ban91] suggested that upright handwriting should have predominantly horizontal and vertical lines. He detected peaks in the histogram of line segment angles and used the deviation of these peaks from the vertical as an indication of the character's slant. Güdesen [Güd76] claims the best way to determine inclination is to calculate the moments of inertia of the character and from these determine the inclination of the principal axes of inertia. The rotation required for slant correction is that which is needed to make the axis that has minimum moment become vertical.

Once the required rotation has been calculated, the character can be rotated using a simple transformation. Problems arise if the character is placed on a discrete grid such as a bitmap. Rounding errors in the new positions of black pixels can be considerable, causing breaks in the character and substantial structural changes [Güd76]. Güdesen found that the classification error rate increased by a factor of four in his experiments with rotation normalization on bitmaps. For character representations with continuous coordinates, such as outlines, these quantization effects do not occur and rotation is therefore a reasonable alternative to the more popular shearing technique.

Shearing is a less computationally expensive way of correcting slant that performs a simple transformation of character coordinates in one direction, usually the x direction. Figure 2.2i gives an example of slant correction by shearing. In this example the required correction is estimated from the ascender of the character. This results in the ascender in the upper half of the bitmap becoming precisely vertical. Note however that the equal but opposite shearing of rows in the lower half causes some undesirable distortion in this example.

The amount of shearing is usually determined by some form of moment computation, as with rotation. Moments may be used to estimate the angle change required; the shear transformation is then determined based on this angle. Alternatively, the moments themselves are used as part of the transformation matrix, the elements of the matrix being set so that the operation will zeroize certain moments (see section 2.5.1 for a definition of moments). The latter method was first proposed by Bakis *et al.* [BHN68] for zeroizing the $m(1,1)$ moment. It has since been used by several other researchers, e.g., Naylor [Nay71]. A variant of the former method was used by Bozinovic and Srihari [BS89]. They determined the slant of words by averaging the slopes of the connecting lines between the centres of gravity of the upper and lower halves of specific portions of the image. They do not state how they calculated the centre of gravity but moments are a common method for determining this point. The specific portions of the image used were the line parts of the image (as opposed to junction parts which are not good indicators of slant). A complicated, though reasonably effective, series of operations was used to remove certain pixel rows from consideration, such as those rows with long runs of black pixels. Afterwards, only the thin line parts of the image remain. A shearing of x coordinates was then applied to correct the average slant of the word. The limitation of their method is that it requires examination of the whole word to determine an accurate estimate of inclination. It is most effective on words with consistent character slants.

Other methods of slant correction have been used. Dutta [Dut74] based skew normalization on the ratio of "multiple" to "non-multiple" pixel columns. Multiple columns of a bitmap are defined as those where the run of pixel colour changes more than twice, e.g., 000110011000 is multiple as it has four changes of pixel colour. Non-multiple columns are those with two or fewer colour changes. Dutta tried twenty-four different inclinations of words and selected the one with the greatest number of non-multiple columns (based on the assumption that this is the inclination where the most lines run vertically). This criterion was effective on Dutta's data which was very clean but it would be highly susceptible to noise in a practical application. Another weakness of this method is that the use of twenty-four different transformations is computationally expensive compared to methods which estimate the desired change in slant prior to transformation.

Casey [Cas70] corrected slant with a moment normalizing transform similar to that of Bakis *et al.* but shearing both the x and y coordinates. The $m(1,1)$ moment (see section 2.5.1) was zeroized as in Bakis *et al*'s method and also the $m(0,2)$ and $m(2,0)$ moments were equalized to an arbitrary constant. Nagy and Tuong [NT70] used an alternative technique to normalize slant in both the x and y directions. From the convex hull of a character they determined four points called *fiducial marks*. These are the four points where the slope of the sides of the convex hull turn through the intercardinal directions (45°, 135°, 225° and 315°). The quadrilateral formed by the fiducial marks is spread so that it encloses the character. This quadrilateral is transformed to a square (or rectangle) using a geometric projection. This involves projecting the quadrilateral from a focal point onto a plane; any quadrilateral can be transformed to a square in this way. This method produced large distortions in character where two fiducial marks occur close together, such as in '7's and some '4's. Nagy and Tuong dealt with these characters by enclosing them in triangles instead of quadrilaterals and then enclosing the triangles in parallelograms, upon which the projection was performed.

Words in a document that do not run horizontally present a problem to segmentation (particularly to simple vertical line segmentation) and increase the variability of character slant, so it is desirable to align all words horizontally. One strategy for this is to shear or project correct individual characters in the y direction as well as the x direction using methods such as Casey's and Nagy and Tuong's. The other is a normalization technique, known as *slope correction*, which re-orientates whole words or text lines so that they run horizontally. This may be done by estimating the baseline of each word (or text line), finding the difference in angle between the baseline and the horizontal and performing a rotation to re-orientate the word. Slope correction is less commonly used than slant correction because in most applications a careful presentation of the document to the scanner will keep line slope deviation to a minimum.

Line-width normalization is a term for methods which adjust the thicknesses of strokes. Strictly speaking, thinning characters to unit pixel width, as described in sections 2.3.1.1 and 2.3.1.2, is such a method; however, the term is generally only used to refer to methods which normalize lines to something other than unit pixel width.

Line-width normalization is a rarely used approach that arose as an early reaction to thinning techniques. The early use of thinning was not intended to facilitate higher level feature extraction but simply to make similar characters more alike, thereby aiding the relatively simple classification methods that had previously been applied to unthinned characters. The features used were generally pixel-based rather than stroke-based and so did not benefit from the thinned representations other than from the uniform thicknesses of lines.

Many of the early researchers found, however, that the unit-width lines actually had an adverse effect on recognition accuracy. Analysis of the statistical parameters of the features showed that thinning of the characters decreased the distance between the mean feature vectors of character classes and greatly increased the variances of those classes [Güd76]. Discrimination of thinned images is therefore much more difficult if based solely on their pixel distributions. The loss of accuracy was also associated with the reduction in the number of black pixels in the bitmap. The reaction to this was a type of line-width normalization that preserved or increased the number of black pixels in the image, rather than reducing it.

Güdesen [Güd76] used a thickening technique that transposed the bitmap representation onto an identical copy of itself but shifted one pixel in one of four directions. This did not truly "normalize" the line thicknesses, it merely increased them by one pixel width. While this smearing of the characters risks destroying some detail, such as small holes or concavities, it was found that transposing in the downward direction reduced the error rate by approximately 30%. An example of this technique is shown in figure 2.2j.

Genchi *et al.* [GMW68] used another kind of line-width normalization on numeral data taken from envelopes. They found that stroke width varied from 0.2mm (for scratchy ballpoint pens) to 1.0mm (for worn felt-tips). Using a scanner resolution of 5 pixels per millimetre gave them strokes between one and five pixels thick. After vertical scaling (size normalization) they used a 3x4 mask for iterative thinning. However, rather than repeatedly apply the pixel erosion until a complete skeleton remained, they applied only one erosion pass. Provided the original width was not more than five pixels, this single pass was sufficient to thin the strokes to at most two pixels thick (a criterion required by their feature extraction process).

Line-width alteration is one of the most distorting normalization techniques. Neither of the above systems have a precise definition of what the normalized line width should be and so cannot truly be called "normalization." In fact, it is theoretically possible that the process could cause a character's stroke widths to vary more than they did originally. Although Güdesen showed that smearing could improve classification based on pixel distributions, it seems likely that it would eliminate detail required by other types of features such as outline-based transforms, geometric and topological features. Semi-thinning algorithms such as Genchi *et al.*'s have limited usefulness; normally systems either require that a character be completely thinned or require no thinning at all. These techniques are therefore rarely ever used.

2.4. Segmentation

Segmentation, the task of separating text into its component characters, is one of the most difficult and fundamental problems in character recognition. While a reasonably high degree of success has been attained on machine-printed characters, an effective algorithm for segmenting handwritten characters, even simply connected ones, has yet to appear.

Segmentation of text into words or numeral strings can be performed with reasonable accuracy. Long horizontal runs of white pixels indicate divisions between lines; stretches of vertical runs of white pixels within lines indicate gaps between words. There are still complications when words touch but research has focused on the more difficult problem of segmenting characters within words (the idea being that, if successful, techniques for character segmentation will also be applicable to word segmentation).

The difficulty arises from the overlapping, touching and splitting of characters. Rules for deciding when and how to divide a connected component (a black pixel region) between multiple characters and when to group multiple components into a single character are extremely difficult, if not impossible, to generalize.

An alternative approach is to treat words as entities and attempt to recognize them, without segmentation, from the relative positioning of identifiable features, usually ascenders, descenders and holes, within the word, e.g., Hull and Srihari [HS86], O'Hair and Kabrisky [OK91]. This approach is limited to applications where only a small lexicon of words needs to be learned by the recognizer. In a large lexicon, such as a whole dictionary, the number of possible word shapes requires huge sets of exemplar samples for classifier training; such techniques are generally limited to template matching classification. Even then, there are many words with very similar shapes, e.g., "chewing", "drawing", "droning", "shaming", "sharing", "shaving", "shoring", "shoving" and "showing". For shorter words the similarity is even greater. It is also difficult to see how this approach would work for numeral strings.

Many approaches attempt a single-pass character segmentation. For example, Tsuji and Asai [TA84] formulated segmentation of machine-printed characters as two problems: pitch estimation and character sectioning decision. They used a statistical analysis method based on least square error to estimate pitch and a dynamic programming method with minimal variance criterion to select vertical character sectioning lines from a candidate set. Pervez and Suen [PS82] used a simple pitch-based method to segment, with high accuracy, a database of 1030 unconstrained, handwritten, binarized, numeric ZIP codes. Their method relied on knowing the precise number of digits in the ZIP code. Hull *et al.* [HSC88] used a similar approach. Shridhar and Badreldin [SB86] [SB87] analysed pixel profiles to group well-spaced but broken handwritten numerals and used a "Hit and Deflect Strategy" to perform nonlinear segmentation of simply connected numerals. Hypotheses about the connectivity of numeral strings (whether the string is an isolated character, two overlapping but not touching characters, or two merged characters) were tested using a decision tree before segmentation and recognition were attempted. They also required to know the exact number of numerals in the strings. Kimura and Shridhar used a slant splitting algorithm based on discriminant analysis instead of the "Hit and Deflect Strategy" [KS91] and extended the previous work to split strings of any length [KS92].

Single-pass methods that attempt to select the correct segmentation with no information as to what the character is do not work sufficiently well even on numeral strings. Recognition of the characters and contextual consideration of the strings is essential to accurate segmentation.

Shlien and Kubota [SK86] represented a line's pixel columns as vectors and quantized them into one of eighty possible encodings (called prototype codes). String matching recognition was applied to the sequence of prototype codes to pick out possible characters and their positions. Sequences of characters in compatible positions were then dictionary checked to find a contextually sensible sequence. Segmentation is only by vertical divisions and relies on being able to recognize characters from the prototype encodings. These encodings were effective for machine-print (though the overall method was not particularly successful) but are unlikely to tolerate the huge style variation of handwritten characters.

Kahan *et al.* [BKP86] [KPB87] identified merged machine-printed characters based on recognition by a probabilistic character classifier. If, for each character, the classifier's probability of the target being that character is less than the *a priori* probability of it being that character, then the target is considered to be merged. A vertical segmentation line is found from the maximum peak in the ratio of the second difference of the pixel projection to the pixel projection itself (this ratio was used, rather than just the second difference, to bias the segmentation line towards thin areas of the target). The method performs well on lightly touching characters but failed to segment heavily touching characters, particularly those where the pixel projections vary gradually due to a lack of vertical strokes near the merge; for example, "oo", "oa", "od" and "oe".

In 1982 Casey and Nagy [CN82] proposed the principle of recursively segmenting and classifying characters. Where the initial segmentation yields a low certainty character classification, the line is re-segmented and reclassified. The process iterates until an acceptable certainty factor is achieved. Using vertical line segmentation, Casey and Nagy successfully resolved approximately 85% of merged characters in a small set of variable-pitch machine-printed characters.

Leedham and Friday [LF] used a recursive approach to segmenting handwritten upper case characters. They noted that the commonly used vertical line segmentation cannot easily isolate certain upper case characters: 'A', 'J', 'S', 'T', 'V', 'X', 'Y' and 'Z'. Therefore, rather than using the usual vertical pixel projections, angular pixel projections were used instead. 80% correct segmentation was achieved on 128 samples from the Essex database of address images (see section 2.6.3).

Tsujimoto and Asada [TA91] segmented machine-printed text using recursive segmentation and recognition to construct a decision tree of candidates for break points and their resultant acceptable character candidates. Linguistic context was used to select the correct characters and break points. Knowledge about omni-fonts was used to determine when a character candidate is to be accepted, rejected or when it should be accepted but remains ambiguous (e.g, an 'l' may be accepted but could also be an 'I', '1' or 'i'). Knowledge about character composition (e.g., an 'm' is like an 'r' next to an 'n') was used to aid the recursive segmentation and recognition. The method performed well although their vertical line segmentation produced some failures on the machine-printed text; it is likely that handwritten text would produce many more. Incorporating more complex splitting would increase the size of the decision tree, possibly making it too slow for practical use.

Liang *et al.* [LSA94] implemented a method for recursive vertical line splitting or merging of machine-printed text, directed by classification acceptance at each stage. Their method is much faster than Tsujimoto and Asada's. Vertical line break points are determined by discriminant functions based on the pixel and profile projections (the use of profile projection is to overcome the problems with Kahan *et al.*'s method). Knowledge of a character's ascenders and descenders is used to correct errors in classification. Knowledge of character composition aids the merging process. Spell checking is used to confirm recognition and remove ambiguity. A correct recognition rate of 99.65% was achieved on 54,000 characters segmented from a reasonable quality machine-printed document; only 185 segmentation errors occurred.

Little work has been published on segmentation of cursive script. The main work on the subject is by Bozinovic and Srihari [BS89]. They segment using vertical lines at potential segmentation points determined by detecting local minima along the lower edges of the word outlines and areas with low vertical profiles. This vertical segmentation is aided by slant correcting the words prior to segmentation, but they note that there are still problems such as large loops or strokes overshadowing adjacent letters, and t-slashes connecting through letters. Letter classification and dictionary checking is used to determine the correct segmentation points.

2.5. Feature Extraction

Of primary importance to the recognition of characters is the extraction of features which capture the distinctive characteristics of character classes. These features form the inputs to the classifiers which must recognize the characters. Finding suitable features has been the subject of much research in the field.

An ideal feature should be able to:

- distinguish each class from all the others (accuracy)

- capture the distinctive characteristics of each class (generalization)
- reduce the dimensionality of the data
- be quickly extracted
- be invariant to size, position and rotation of the character
- tolerate noise, distortion, and style variation
- tolerate broken or multi-part images
- be translated back to the shape of the original character
- be calculated incrementally as part of an adaptive segmentation system (see section 5.8)

A feature which can achieve all these goals satisfactorily has yet to appear and new features continue to be developed. The vast number of features in the literature prohibits description of them all; however, they can be grouped into a smaller number of paradigms. Representative features from each paradigm are reviewed below.

Suen [Sue82a] divides feature detection schemes for character recognition into two families: global analysis and structural analysis. Global analysis techniques include point distribution measurements, global transforms and physical measurements. Structural analysis produces geometric and topological features.

2.5.1. Point Distribution Measurements

Point distribution methods generally work from pixel-based representations. The most basic form is simply the bitmap representation itself, passed to a template matching classification system (see section 6.3.1). Such an approach has been used by Shimura [Shi73] and Miloslavskaya and Polyakov [MP69a]. A hardware template matching correlator was implemented by Kozlay [Koz71]. This approach is extremely slow since it does not reduce dimensionality; neither does it achieve high levels of accuracy. Point distribution methods have been extended to produce real features in several ways: *n*-tuple methods, zoning, moments, characteristic loci, crossings and distances.

Rather than using all the points in a bitmap, *n-tuple* methods take a subset of points and use their binary states (black or white pixel values) as features. The point subset may be single pixels, pairs, triplets or groups of n pixels. These are then matched against a template set. This method was proposed by Bledsoe and Browning [BB59] in 1959 and has since been used by several other researchers, e.g., Bakis *et al.* [BHN68], Ullmann [Ull69] and Kwan *et al.* [KPS79]. Austin [Aus88] extended the method to greyscale n-tuples.

Zoning is the division of an image into several regions, possibly overlapping, and recording the black pixel density in each region as a feature. Pixel density provides useful information such as centre of gravity and symmetry of a character. Hussain *et al.* [HTD72] divided character frames into a 5x5 grid and used the density of each of the 25 squares as features. Breuer and Vajta, Jr [BV75] measured point distribution projections in horizontal and vertical zones.

A widely used feature is a set of *moments*. Moments, developed for character recognition by Hu [Hu62], are formed by summing the distances from each black pixel to a reference point or line. Hu also presented moment invariants (later corrected by Reiss [Rei91]) which are functions of the moments that are invariant under certain transformations of the image: rotation, scale and skew. The standard two-dimensional moments of an image function, $f(x,y)$ are defined to be

$$m(p,q) = \sum_{x=1}^{M} \sum_{y=1}^{N} f(x,y) x^p y^q$$

where $M \times N$ is the size of the image, x and y are the row and column coordinate in the pixel matrix, and $p+q$ is the order of the moment. Position invariant moments can be formed using the centroid, $(\bar{x},\bar{y})$, of the character image as the reference point:

$$m(p,q) = \sum_{x=1}^{M} \sum_{y=1}^{N} f(x,y)(x-\bar{x})^p (y-\bar{y})^q$$

Moments up to order five were used by Spanjersberg [Spa74]; Kwan *et al.* [KPS79] used up to sixth order moments. An alternative to taking moments from the whole pixel matrix is to take them only from the boundary pixels of a character. This approach was compared favourably with the former, silhouette-based moments, by Belkasim *et al.* [BSA91]. Sardana [Sar93] developed the boundary-based moments and applied his Edge Standard Moment and Edge Moment Invariant features to a wide variety of image recognition problems including handwritten character recognition.

A feature system called *characteristic loci* was used by Knoll [Kno69] and Glucksman [Glu71]. Each white pixel in a character bitmap generates a four digit code. Each digit in the code counts the number of crossings of black pixel regions made by a vector extended from the white pixel in one of four directions (up, down, left and right). The number of crossings counted was limited to two to limit the dimensionality of the features. These codes are compared to those of a template set. Similar crossing counting methods were used by Kwon and Lai [KL76] and by Calvert [Cal70], who only used horizontal crossings. Suen [Sue82b] extended the characteristic loci method to multidirectional vectors.

Crossing methods have been extended to include distance to the crossing (Doyle [Doy60]) and length of the crossing (Ni *et al.* [NDG80]). Some methods use only the distance measurements, e.g., Troxel's variation on characteristic loci [Tro76]. Distance measurements from fixed points on the character frame have also been used. Marill and Green [MG60] and Chen [Che65] placed characters on a square grid and measured the closest distance of a point on the character to eight reference points on the edge of the grid (the four corners and the four midpoints of each side), along a line through the centre of the square. The measurements were taken manually by counting squares on the grid. Fu *et al.* [FCC67] used the same method but used a circular frame and measured distances precisely.

Point distribution methods are generally fast and simple, and achieve some tolerance to noise, distortion and style variation [SBM80]. Characteristic loci, crossing and distance methods show some ability to capture the characteristics of a class's shape. However, existing methods have not been invariant to size, position and rotation.

2.5.2. Global Transformations

Global transformations of character image representations reduce the dimensionality of the data and at the same time can provide invariance to certain global deformations such as position, size and rotation. These transformations convert the character representation to a series, spectrum or vector. The transformations may operate on pixel level representations or may be applied to closed curve boundaries (outlines). Outline-based transforms have so far been limited to only one closed loop. This limits their effectiveness at discriminating multi-outline characters such as 'i' and 'j', and broken characters, such as typically occur in practical, noisy applications. The most common types of global transform features are the

Karhunen-Loeve expansion, Hough transform and Fourier and Walsh descriptors.

As it is applied to character recognition, the Karhunen-Loeve expansion [Wat65], also known as the principal component or Hotelling transform, is based on diagonalization, by extraction of eigenvectors, of the covariance matrix of the binary character representation. This series expansion has been used in many experiments including Krause *et al.* [KSP74], Ott [Ott74], Güdesen [Güd76] and Niemann [Nie77].

Fourier descriptors [Cos60] use the Fourier series expansion to encode a single character outline as an infinite sum of sine and cosine terms of different frequencies, phases and amplitudes. The coefficients of the frequency-ordered terms give the encoding of the character (the feature values). There are other ways of interpreting the series but intuitively the sine and cosine interpretation is probably the simplest. Extraction of Fourier coefficients is usually performed using Cooley and Tukey's improved algorithm, the Fast Fourier Transform [CT65].

Fourier descriptors make use of the similarity of Fourier power spectra between closed curves which vary only in size, orientation and position to give features for shape discrimination which are invariant under these transformations [ZR72] [LS81]. However, the Fourier power spectra alone are often not sufficiently unique for accurate character discrimination [CL88]. Tests on the CEDAR upper case character data, using the first 20 Fourier amplitude coefficients, showed very poor discriminatory ability on the handwritten characters (correct classification rate was only 44.16% on the training set and 36.52% on the test set). Persoon and Fu [PF77] suggest that Fourier descriptors are useful for shape recognition problems where "classes can be described by a few fixed prototypes that may be rotated, translated, scaled, and that can possess some (random) noise on the boundaries." However, handwritten character recognition requires far more than a few fixed prototypes. Despite this, Fourier descriptors have been popular in the field and have been used by many researchers, e.g., Granlund [Gra72], Zahn and Roskies [ZR72], and Krzyzak *et al.* [KLS89]. Other work has aimed at improving results by combining Fourier descriptors with other features. Lai and Suen [LS81] used a combination of 10 Fourier descriptors and 42 boundary encoding features to achieve high recognition accuracy on a small, highly constrained, handwritten character set. Shridhar and Badreldin [SB84] used topological features in addition to Fourier descriptors.

Walsh descriptors are the coefficients of a similar series expansion to the Fourier transform but based on the normal orthogonal functions of Walsh [Wal23]. It has been used on polygonal curves, e.g., Sarvarayudu and Sethi [SS83], and on pixel-based character representations, e.g., Huang and Chung [HC87], and Rajavelu *et al.* [RMS89]. While good recognition rates have been achieved on machine-printed characters, the Walsh transform has not proved any more effective on handwritten characters than the Fourier transform.

Other similar transforms that have been used for character recognition (often in combination with the standard Fourier transform) are the Mellin transform (Casasent and Psaltis [CP76]), Hadamard series expansion (Wendling and Stamon [WS76]), Haar expansion (Wendling and Gagneaux [WG77]), projection transform (Li and Cheng [LC83]) chain-code transform (Cheng and Leung [CL85]), Hartley transform (see [Low91]) and Gabor wavelets [Shu94].

There are several problems with global transforms based on series expansions. While the full, infinite series expansions may contain all the information in the image, the finite subset of the first *n* terms may not. Valuable information may be lost in the uncalculated terms. Although it may be possible to recreate the original character image very accurately from an expansion, it is difficult to relate the coefficients of a series to distinctive characteristics of a class. The series expansions are encodings of the character but are not, strictly speaking, features. While the transforms can be shown to be invariant to properties such as scale, rotation and translation, there is no reason to believe they are tolerant to variations in style. In fact, these types of global transformations have only proved to be successful features for machine-printed characters, whose styles vary much less than handwritten characters, and for characters handwritten by trained authors, according to standard models [LS81]. Another problem common to most series expansions is the difficulty in determining an analogous starting point for each case. Small variations in starting point can produce very large differences in the coefficients. Finally, the calculations needed to extract global transforms are often computationally expensive.

One type of global transform that is not based on a series expansion was patented in 1962 by Hough [Hou62]. The Hough transform is an efficient template matching method for detecting curves in noisy digital images. Duda and Hart [DH72] extended it beyond its initial two-dimensional form to function in three-dimensional space. In a landmark paper in 1981 Ballard [Bal81] developed Merlin and Farber's early attempt at generalization [MF75] to detect arbitrary shapes at any orientation or scale. Davis [Dav82] extended the generalized Hough transform further to the matching of hierarchically organized point patterns and the matching of geometric objects (line segments, circular arcs, etc.) rather than points. The Hough transform exploits the duality of points on a curve and the parameters of that curve [Bal81]. It maps a pixel-based image representation to a parameter space and detects curves by examining the clustering in that space. The difficult global detection task is transformed to a much simpler local peak detection in parameter space. The advantages of the transform are that it is invariant to rotation, position and size, it can be computed in parallel, it can recognize partial or occluded shapes and it is very robust to noise. The main disadvantage is that each new shape to be recognized requires a specific Hough transform to detect that particular shape. Images consisting of different shapes therefore require several successive transforms. Another disadvantage is that the storage and computation requirements are large; however, much work has focused on improving the transform's efficiency.

The Hough transform has been used extensively in the general field of image analysis. Illingworth and Kittler give a comprehensive review [IK88]. Its use in character recognition has been less common though it is rapidly growing in popularity. Kushnir *et al.* [KAM83] [KAM85] used the peaks in a Hough transform line detection parameter space as features in a Hebrew character recognizer. Hebrew characters consist almost entirely of linear strokes so the line detection Hough transform was well suited to the task; a recognition rate of 86.9% was reported on handprinted characters. Oulamara and Duvernoy [OD88] used Hough transform features in the recognition of Berber characters. More recent work has been in producing a probabilistic Hough transform [KEB91] [Ste91] which is more useful in character recognition where probability-based candidate sets are commonly required as output (to be subsequently input to a contextual postprocessor).

The Radon transform (Radon [Rad17], see also Deans [Dea83]), of which the Hough transform is a special case [Dea81], has also been used for character recognition. The advantage of the Radon transform over the Hough transform is that it can detect any analytically defined shapes, without requiring separate instances of the transform for each shape; different shapes' distributions in the transform space are "relatively transparent" to each other [LB87]. Leavers and Sandler [LS88b] developed an efficient Radon transform using methods from Ballard's generalization of the Hough transform [Bal81] and table-lookup to speed computation. This was applied to character recognition by Leavers [Lea90].

2.5.3. Physical Measurements

Physical measurements in Suen's taxonomy are predominantly the height and width of a character. These are commonly used when size normalizing characters (see section 2.3.5) prior to further feature extraction. Other physical measurements have also been used, notably the height and width of specific areas of characters to distinguish ambiguous characters. Duda [Dud70] measured the maximum widths of the top and bottom halves of characters and used their ratio to distinguish the characters '8' and 'B'. Suen and Shillman [SS77] used physical measurements such as width ratio of the left and right "limbs" of characters to distinguish 'U' and 'V'. Shridhar and Badreldin [SB86] used height-width ratio and character widths on each scan line as physical features. Other physical measurements that have been used are counts of the number of connected components in a character and number of interior holes. These features are not sufficient to discriminate large character sets on their own and must be used in conjunction with another type of feature.

2.5.4. Geometric and Topological Features

The most popular type of features are those which aim to detect significant geometric or topological properties of characters. These provide a description of the makeup of characters which is more comprehensible to human researchers and can more easily be related to human abstractions of the distinctive features of characters. If such features can be accurately detected and characters match the human abstractions, this method should exhibit high tolerance to style variation and distortion. It has therefore been the focus of much investigation.

Geometric and topological features are extracted from a line-based representation of the character such as the outline or centre line (produced by thinning or vectorization). Pixel-based formats are sometimes converted to a line-based form by edge detection, which assigns each edge pixel a direction of slope (usually by a masking technique, e.g., Genchi *et al.* [GMW68]). This is to preserve detail of the slope of lines which is not readily accessible in the usual orthogonal outline representation. Thinned or vectorized representations are clearly the most suitable for this type of extraction, particularly when represented in stroke-based or stroke-end-based forms.

Once a line representation has been obtained the geometric and topological features are extracted. These are usually related to position in the character so they may be compared with other cases, though simpler strategies have been used such as simply counting the number of occurrences of particular features, e.g., Huang and Chuang [HC86], Kittler [Kit80]. Common features used are listed in table 2.1.

Feature	Related Measurements	Example References
Lines (strokes)	orientation, length, curvature	[GMW68] [TG72] [AP77] [KPS79] [MH83] [Bai88] [BK88] [Shl88]
Bays (concavities, convexities, arcs, hooks, cusps, bends)	orientation, area, curvature, depth	[BHN68] [TG72] [AP77] [KPS79] [NDG80] [YM80] [Bai88] [MG89]
Corners	angles	[MH83] [BK88]
Endpoints (line tips, terminals)	approach angle, width	[Kuh 63] [Hos72] [FKH76] [HLN80] [MH83] [Bai88] [BK88]
Junctions (intersections, forks, branches, nodes)	number of branches	[GMW68] [KPS79] [YM80] [Kit80] [HLN80] [MH83] [Bai88] [BK88]
Loops (holes, lakes, islands)	area, radius, perimeter	[Spa74] [AP77] [KPS79] [MH83] [YM80] [Bai88]
Splits (junction boundaries)		[Hos72]
Spikes (protrusions, intrusions, sharp angles)	orientation	[Kuh63] [AP77]
Points of Inflexion		[TD70] [YM78]

Table 2.1 Commonly used geometric and topological features.

Some research has used polygonal approximations (see section 2.3.2) to simplify geometric and topological feature extraction from outlines, e.g., Mantas and Heaton [MH83]. In some cases the approximations themselves have been used as feature strings. Tsai and Yu [TY85], Tsay and Tsai [TT89], and Maes [Mae91] have all applied string matching techniques to feature strings constructed from linear polygonal approximations (straight lines), but their techniques have so far been limited to recognizing relatively simple shapes. These techniques are discussed in more detail in section 6.3.2. Accurate recognition of character shapes using this approach seems an unlikely prospect.

As with series expansions (see section 2.5.2), polygonal approximations are an encoding of the characters but do not obviously detect the distinctive characteristics of character classes. Although they are more likely to capture these characteristics than series expansions, the huge style variation of handwritten characters means that line segments in corresponding positions in the approximations of similar characters may bear little or no relation. They are also hampered by noise and, particularly, breaks in outlines. Polygonal approximation strings can only describe a single connected component so broken and multi-outline characters cannot be easily handled. Similar problems affect other types of geometric and topological features.

For machine-printed characters, geometric and topological features can be fairly tolerant to style variation, rotation and translation, as the characters will usually conform to the expected topology. Handwritten characters, however, do not obey strict rules of geometrical grammars in practice. Strokes which are expected to cross may stop short so that junctions will not be formed. In cursive script, lines may not terminate at expected endpoints but may continue on to join the next character, crossing the target character on the way and producing unexpected junctions. This makes certain features difficult to detect accurately. Topological features are therefore highly variable for similar handwritten characters. Geometric features are also very prone to noise. Noise may cause spurs (which become incorrectly detected as endpoints), joins (incorrect junctions), breaks (causing features to go undetected) or more general distortions in the detected line sections.

2.6. Databases for Character Recognition

A major difficulty in character recognition research is the comparison of methods. The diversity of testing procedures is one cause of this but the main reason is the use of different test data by different researchers. A standard database is required for testing if a meaningful comparison of techniques is to be made.

In addition to allowing comparison of data, the availability of large character databases saves a great deal of time for researchers who would otherwise have to gather data themselves. In particular, segmentation of words into characters by hand is a common requirement (where segmentation is not a research aim) and a very time consuming process. Pre-segmented databases are therefore of great value.

The only disadvantages to standard databases are that certain assumptions may be made in the gathering, scanning and storing of data which may not be shared by the researcher. It may also be difficult to replicate these assumptions if a researcher wishes to gather new data. Furthermore, the hand processing and pre-segmenting of data may be drawing researchers' attention away from these early stages of recognition which are perhaps the most important areas on which to concentrate. Many published approaches overlook the need for recognition techniques to be integrated with a recursive segmentation system; the availability of data in a pre-segmented form has led people to believe that segmentation is an independent process.

Despite the availability of large databases and the advantages of standardization, some researchers still prefer to create their own. The problem of comparison of results remains. However, the best research generally makes use of recognized, standard test databases to demonstrate the credibility of its results. Such databases are steadily growing in popularity and new databases continue to appear, to keep up with advancements in scanner technology and recognition performance.

Many different databases have been made available to researchers in the field. The following two sections describe in detail the databases used in this research, and the third outlines other standard databases.

2.6.1. NIST Special Database 3: Handwritten Segmented Characters

The National Institute of Standards and Technology (NIST) Special Database 3 [GW92] contains binary images of handwritten alphanumeric characters. 2100 form images, scanned at 12 pixels per millimetre, are included along with the characters contained in those forms. The characters have been segmented and stored individually in 128x128 pixel images. The segmentation was manually checked and NIST report a less than 0.1% error rate between the segmented character files and the associated character class. Characters are divided into sections of 223125 digits, 44951 upper case and 45313 lower case.

The NIST images are in a bitmap variant format, binary raster stream format (a left to right, top to bottom list of the binary pixel values). A header is included which stores additional information such as height, width and offset of the image. These are then stored in a Multiple Image Set (MIS) file format which stacks several character images of the same dimensions in a single file to reduce directory sizes. The files are two-dimensionally compressed using CCITT Group 4 compression [CCI84]. A suite of programs are provided for the extraction of images from the database.

Each character class is well represented in the database with approximately equal numbers of each class within each section (digit, upper and lower case). There are roughly 22000 samples of each digit, and 1700 of each upper and lower case character. The quality of the images is fair with low levels of noise. A wide variety of styles, pen thicknesses, slopes and slants are present though, making this database a difficult challenge for character recognition.

2.6.2. CEDAR CDROM 1: Database of Handwritten Images

The Center of Excellence for Document Analysis and Recognition (CEDAR) CDROM 1 database [Hul94] is an image database of alphanumeric characters from handwritten mailpieces (envelopes). The images were gathered from real mail addresses at the main post office in Buffalo, New York, as part of a research project sponsored by the United States Postal Services (USPS) Office of Advanced Technology. A high quality flat bed digitizer, the Eikonix EC850 4096x4096 CCD, was used to scan the addresses in 8-bit greyscale at 300 pixels per inch (approximately 12 pixels per millimetre).

The database stores roughly 5000 city names, 5000 state names and 10000 ZIP codes. In addition, approximately 50000 alphanumeric characters are segmented and stored individually in both the original greyscale and in bi-tonal form (obtained by thresholding the corresponding greyscale image). The bi-tonal images are divided into a training and a test set, with roughly 90% of the characters forming the training set and 10% forming the test set. Each set is further divided into an alphanumeric set and a numeric only set.

The character images are stored, one per file, in HIPS format [LCS84]: a printable ASCII header containing descriptive information such as image height and width, followed by a header terminator sequence (Carriage Return, '.', Carriage Return), followed by a one byte per pixel raster stream of either the greyscale or bi-tonal data. The bi-tonal images are bit-packed to one bit per pixel. The HIPS file is then compressed using 'delta' compression — a public domain difference coding technique for lossless compression. C source code is provided for the extraction of images to HIPS format and conversion to PBM format.

The pre-definition of training and test sets is a sensible move to reduce the difficulty of comparing the results of different researchers, who commonly use different test data despite using the same databases. The use of real life data, unconstrained by author, writing style or writing implement, makes this data set of great use for evaluating practical performance of recognition systems. Laboratory generated character images are biased in that the authors are aware their writing will be used for recognition purposes and may give abnormally neat or abnormally sloppy samples in an attempt to either achieve high recognition results or to "fool the computer."

The main weakness of the database is the small number of alphabetic characters. In particular, the infrequently occurring characters 'Q', 'Z', 'j', 'q' and 'z' are very poorly represented (there is only one 'j' in the training set). This makes learning of the full set of upper or lower case characters impractical; much larger sample sets are required. Another problem with the segmented characters is that only the largest connected component has been stored (by the human segmenter) for each character. In the case of 'i's and 'j's the dot is left out. This may suit the many recognition systems which cannot handle multi-part images but it distorts the data. It also makes it impossible to distinguish between 'i' and 'l'. No broken images are present in the database so a recognition system's tolerance to such breaks cannot be tested. The quality of images is fair. There is some noise on the edge of the characters but the extraction of only a single component eliminates background noise.

2.6.3. Other Standard Databases

Traditionally the most widely used databases in the field have been those of Highleyman, Munson and Suen *et al.*, made available by the IEEE.

Munson's database [Mun68] consisted of 12760 samples of the basic characters of the FORTRAN II alphabet (46 characters). 6762 samples were collected from 49 authors who each wrote three sets of the alphabet on standard general-purpose coding sheets using an HB thin-lead mechanical pencil. The remainder came from a single author or from actual coding sheets. The authors were instructed to write 'Z's and '0's with slashes and put crossbars on the 'I's. The coding sheets were scanned at constant viewing distance with a vidicon television camera using a close-up lens. The generated waveform was quantized to a bi-tonal image with a Schmidt trigger. The 120x120 point raster scan was then scanned for characters, which were isolated and transferred to a 24x24 raster format. The character isolation process allowed for multi-part characters.

The single author and multiple author files were divided into training and test sets. Human isolated character recognition error rates on the test sets averaged 0.7% on the single author set and 4.5% on the multiple author set. A machine recognition rate of 92.98% has been achieved on the numerals in Munson's data. Munson himself reported 97.0% accuracy on the single author data and 85.0% on the multiple author set. Suen *et al.* achieved 84.52% on a small subset of the database.

Highleyman [Hig61] gathered 500 numerals and 1800 upper case letters from 50 authors who were requested to write neatly. Resolution of the data is low and high recognition rates were never achieved on the set. The data does not reflect the resolution obtainable with modern scanner technology and so is unlikely to be used in future research.

Suen *et al.*'s database [SSS76] contained over 100000 alphanumeric characters obtained from 30 authors. The authors were asked to write quickly and carefully in boxes on an OCR form using a thin-lead mechanical pencil. They were asked to write according to 174 model characters. Character models to guide authors have also been used by other researchers including Krause and Bleichrodt [KB73], Caskey and Coates, Jr [CC73] and Mori *et al.* [MMY75].

Of these three main databases, Munson's was the most popular set as it was generally considered to be the most difficult, unconstrained set to recognize [Man86]. The databases have, however, become dated. Modern scanners are capable of greater resolution. The vastly increased size and access speed of affordable disk space means larger image representations can be stored and processed. As a result, higher resolution databases, often in greyscale format, with higher quality scans are now becoming available. Recognition techniques and classifier design have advanced to a stage where tackling real-world problems is an ambitious, but not unreasonable, goal. Modern character databases are therefore much less constrained. Many contain the original, unsegmented document and word images to aid research into segmentation.

Among those likely to be widely used in future research are the NIST databases. The third of these is described in section 2.6.1 but this is only one of many databases for character recognition provided by NIST. The data is unconstrained and presents a realistic challenge to recognition which should make it a popular target.

Another database which has recently achieved popularity among British researchers is the Essex University database of 1549 address block images, scanned from live mail in a British Post Office sorting office [DL90b]. The scanner section of existing AEG OCR equipment was used to generate binary image representations. The mailpieces were constrained by the scanner configuration to be 9" x 4.5" or smaller, with address blocks only in the lower two-thirds of the envelope. Address recognition has useful applications and involves contextual constraints which could potentially produce high levels of accuracy in a practical solution. It is therefore a popular problem with researchers. Much research has focused on the recognition of the postcodes from this database.

2.7. Conclusions

Off-line character recognition is a large, popular and useful research field. It has achieved a level of success sufficient for practical, commercial application on machine-printed documents, but has had only limited success on the more difficult problem of recognizing handwritten characters. Handwritten character recognition has applications in many wide-ranging areas and its enormous potential usefulness and profitability ensures that it will remain an active research field for the foreseeable future.

This chapter has reviewed the current state of the field up to the point where the actual text classification begins. The standard approaches have been described, tracing the commonly used processes from the initial scanning of a document, through its digital representation, preprocessing to a more easily classifiable form and segmentation into individual characters, to the extraction of feature values for presentation to the classifier. In addition, the use of standard databases for training of classifiers and evaluation of system performance has been discussed and popular databases have been reviewed.

At each of the processing stages a wide variety of techniques have been used. Each stage involves some degree of compromise between the requirements of accuracy, robustness, speed and data storage. Different techniques often favour different requirements, hence the diverse nature of approaches. This diversity is also due in part to the lack of a clear winner. Comparison of recognition experiments conducted under different conditions or using different test data is usually meaningless. This difficulty of comparison, and the fact that no approach has yet led to an accurate, robust handwritten character recognition system suitable for general practical use, makes it very difficult to prove the superiority of one technique over another.

The different forms of digital representation provide starting points for different approaches to the character recognition problem. The most common forms are bitmaps, greymaps and outlines. Colour representations are not usually necessary. There is a large diversity in file formats, though they all hold essentially the same information. The field would benefit from greater standardization of these formats.

The effectiveness and worth of preprocessing is discussed in chapter 4. For the moment we can draw a few initial conclusions. Pixel erosion and distance transform techniques for thinning are very sensitive to noise and fail to accurately represent overlapping pen strokes. The vectorization methods, which examine the context of a wider area of the image, can overcome these problems to an extent and therefore appear superior to the former methods. Research, however, continues on each of these approaches. Polygonal approximation is potentially useful for smoothing outlines but is unlikely to prove effective as a pure feature string. The most effective normalization techniques are size normalization and slant correction. Line-width normalization is rarely useful or effective, and thinning can be detrimental if classification is based only on pixel distributions, rather than on higher-level features of the skeletons.

Segmentation of text into individual characters remains a fundamental problem in character recognition. Methods which aim to segment strings correctly on the first attempt are insufficiently accurate as some knowledge about the characters is required. Recursive segmentation and classification is the most promising strategy for the task. Segmentation by vertical lines may be aided by slant normalization but problems still remain with large loops or strokes overshadowing adjacent characters and horizontal strokes extending through neighbouring characters. More flexible divisions between characters will be necessary in a practical system.

A wide variety of features have been proposed for the representation of characters to the classifier. None of the existing features satisfy all the requirements of an ideal feature that were listed in section 2.5. Point distribution methods are fast, simple and show some tolerance to noise, distortion and style. Some types can capture the characteristics of classes to an extent. However, these features have not generally been invariant to size, position or orientation. Invariance to these factors can be achieved with global transformations. These features are merely encodings of characters which reduce the dimensionality of the data needed to store them but do not obviously measure any distinctive qualities of character classes. They are therefore poor features for recognizing characters with a high degree of style variation; they have been effective for machine-printed characters but not handwritten ones. Also global transforms based on series expansions lose information by storing only a finite number of the series coefficients. Physical measurements can be used to discriminate certain characters but must be used in conjunction with another type of feature. Geometric and topological features are partially invariant to position and orientation but the high incidence of gaps between strokes that are expected to touch, or joins of strokes that are expected to be separate, can make some of these features difficult to detect accurately. As with global transforms, the success of geometric and topological features has so far been limited to machine-printed characters. They have yet to achieve high levels of recognition accuracy on the many different styles of handwritten characters.

The main failing of features has been an inability to capture the distinctive characteristics of classes and an inability to tolerate broken or multi-part images. Failure to meet the other requirements of section 2.5 can be partially compensated for by preprocessing or by training the classifier on a large data set, but these two requirements are fundamental to the feature extraction. A new feature will be described in chapter 5 which satisfies almost all of the requirements, and in particular these two important ones.

Standard databases for training and testing of character recognition algorithms are essential if meaningful comparisons of results are to be made. They also save a great deal of time. A number of standard databases are available, two of which have been selected for use in this research.

There is clearly much scope for further research and development in this field. The following three chapters describe advancements made in this area of the character recognition problem and an evaluation of some of the approaches used.

2.8. Nomenclature

$m(p,q)$	Two-dimensional moments of an image function.
n	Level of outline loop.
p	Black boundary pixel for possible deletion in pixel erosion thinning.
w, W	Original and normalized pixel widths in Güdesen's size normalization.

Chapter 3

A New Method for Vectorization of Outlines

3.1. Summary

In this chapter a new method for the vectorization of character images is described which aims to produce an approximation of the pen strokes used in writing the characters. Vectorization is a commonly used preprocessing technique for skeletonization which offers the possibility of improved character representation over the iterative thinning and distance transform methods (see section 2.3.1). A method proposed in 1986 by Theo Pavlidis [Pav86] (and later developed by Elliman [El193]) has been adapted and developed to produce a more accurate vectorization of line segments and a better approximation of pen strokes across junction areas.

Section 3.2 describes the run-length-based vectorization algorithm of Pavlidis and a variation of his approach by Elliman from which the new method was developed. Section 3.3 gives an overview of the method. The determination of pixel edge matches is presented in section 3.4, and section 3.5 explains the rules used in assigning quality scores to these matches. Next, the poor quality matches are removed and line segments are vectorized. Details of this process are given in section 3.6. Afterwards, the special cases of bend and junction areas will remain. The vectorization of these areas is described in sections 3.7 and 3.8. Section 3.9 describes the application of further preprocessing techniques to the vectorized representation such as smoothing and filling. Conclusions are presented in section 3.10 and section 3.11 describes the notation and symbols used in this chapter.

Note that for the handwritten characters on which this research has concentrated, the lines making up a character are pen strokes, but for machine-printed characters the lines are not actually strokes but idealized versions of the conventional strokes used to draw characters. The new method can be used on both types of characters but for ease of description these lines will be referred to in this chapter as pen strokes regardless of whether the image is handwritten or machine-printed.

3.2. Pavlidis's Run-Length-Based Vectorization Algorithm

As discussed in sections 2.3.1.1 and 2.3.1.2, pixel erosion and distance transform methods for skeletonization are prone to noise and do not usefully skeletonize junctions. While they have been successfully applied in some applications, useful feature extraction in character recognition requires that they preserve the structure and shape of characters. Errors caused by noise and an inability to capture structural information when pen strokes overlap has limited their effectiveness in this application. Methods utilizing more global information have arisen to address these problems (see section 2.3.1.3). Most prominent among these vectorization methods is an algorithm proposed in 1986 by Theo Pavlidis, who has previously developed many skeletonization methods.

Pavlidis's run-length-based vectorization algorithm [Pav86] detects groups of similar length run-lengths on adjacent scan lines, which overlap. Some local (height/width ratio) and global rules determine whether the group represents a horizontal or vertical line segment, or a junction area where lines cross each other. Vertical line segments are vectorized by a straight line joining the midpoints of their top and bottom run-lengths. Horizontal line segments are vectorized with a straight line along the bottom of their middle run-length. More complex vectorizations are required for junction areas. These are vectorized according to a local analysis of the angle at which possible vectors cross the junction area. This algorithm has been applied to the recognition of characters in several investigations (Baird *et al.* [BKP86], Kahan *et al.* [KPB87], Baird [Bai88], and Bose and Kuo [BK94]).

Pavlidis's algorithm produces a vectorization in the form of stick-like segments. A connected vectorization which represents the strokes of characters would be significantly more useful. The vertical line vectorizations lose much of the detail of strokes as they use a single straight line across the section. Detail could be preserved by using a chain of vectors joining the midpoints of each run-length in the segment. The horizontal line vectorizations lose even more detail as they are always perfectly horizontal. In many cases, run-length groups representing diagonal lines are identified as being horizontal; the angle of these lines is lost in the vectorization. The junction vectorizations do not consider the angles of line segments approaching the junction area and so are in many cases incorrect. A more complicated analysis of the approaching lines is required to determine the correct path of strokes.

The algorithm only considers pixel runs in the horizontal direction. It therefore requires some very complicated analysis of width changes and colinearity of the run-length centres to determine the allocation of run-lengths to line segment groups or junction groups. It is particularly difficult to distinguish horizontal lines from junction areas. Pavlidis's merging of vectors in non-adjacent groups and "compound vectorization" of junctions are similarly complicated by the unidirectional perspective of his approach.

Elliman [Ell93] proposed a variation on Pavlidis's method based on outlines which looks at pixel runs in the vertical direction as well as horizontal run-lengths. This greatly improves the accuracy of horizontal line vectorizations and removes some of the complications of distinguishing lines from junctions. Elliman used his method successfully to vectorize engineering drawings. The method was developed further by Cripps [Cri94] and Stockley [Sto94]. They modified certain match qualities (see section 3.3) to suit the requirements of engineering drawings, which consist mainly of long, straight lines and arcs. Their modifications are unsuitable for vectorizing handwritten characters where strokes are generally much shorter and more detail must be preserved. New strategies for modifying match quality are needed to tailor the vectorization method to the character recognition problem.

This chapter presents a significant development of Elliman's variation on Pavlidis's method. The outline-based vectorization has been modified and extended to address the problems in Pavlidis's approach. The basic principles of the method — assignment of match quality based on width of pixel runs, removal of matches crossed by shorter ones and vectorization of the simple line segments — are taken from Elliman's method. Original work presented here includes an improved version of Elliman's match removal algorithm, a strategy (suited to character vectorization) for the modification of match quality scores for special circumstances and identification of bend and junction areas, a fast algorithm for approximating approach angles to junctions and methods of junction vectorization based on those angles. The new methods of quality assignment and junction vectorization give considerable improvements over the previous approaches.

3.3. Overview of the Method

The overall strategy of the method is to find the opposite sides of each pen stroke in the given image. Points on opposing sides are paired; we call the line joining a pair a 'match'.

Vectors are formed by joining the midpoints of matches. The main task of the method is to determine which matches provide the best representation of the strokes, and which should be ignored.

Each match is assigned a quality score. This indicates how suitable its midpoint is as a candidate for inclusion in the vectorized image. This quality measure is based on the width of the match. Short matches are generally better than long ones, as illustrated by the simple example in figure 3.1. Assignment of quality scores also depends on the position of the match in the image. This will be explained in section 3.5.

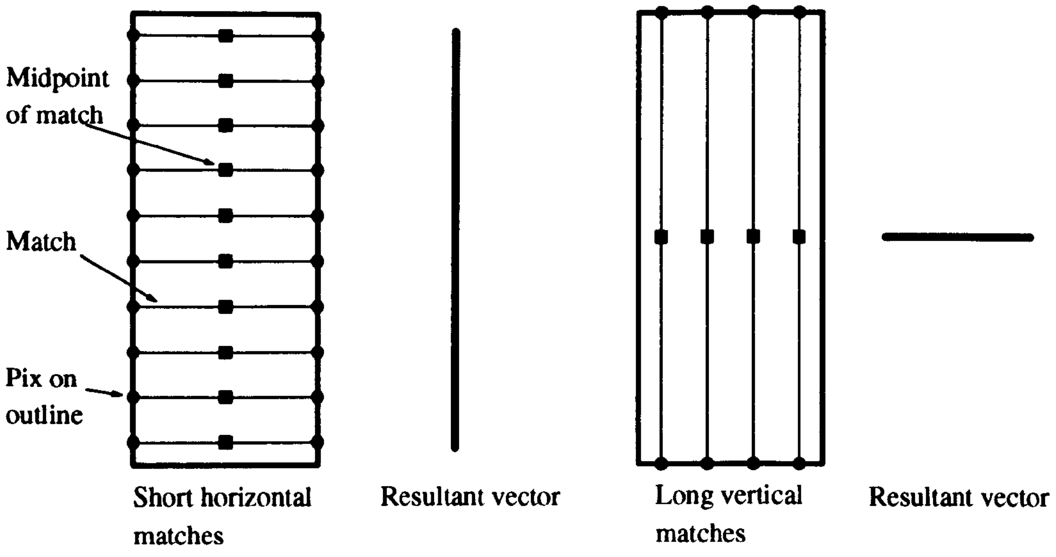


Figure 3.1 A vertical line image and its vector representation from short and long matches.

All the matches will be crossed by others: each horizontal match will be crossed by at least one vertical match and vice versa. It is now required to remove the matches which cross the best quality ones. Low quality scores are best so the matches are worked through in ascending quality order (i.e., lowest score first), removing all crossing matches at each stage. When this process is complete there will be no crossing matches and only the lowest scoring matches will remain.

Where adjacent matches — adjacent in the sense of being next to each other around the outline — are in the same direction, a vector can be created by joining the midpoints of those matches. (This preserves more detail than Pavlidis's method of using a single straight line across sections of adjacent matches.) However, where adjacent matches are in different directions (i.e., one is vertical, the other is horizontal) simply joining the midpoints is not always accurate. These sections of the image are termed 'bends' although they do not necessarily correspond to bends in a stroke.

Another situation where joining the match midpoints is not always the best option is at junctions of strokes. Where two or more pen strokes cross an area, it is desirable to determine the correct path of all strokes through the junction. Joining the match midpoints will only produce a single line and can cause 'bow-ties' (see figure 3.7 for an example). Bow-tie vectorizations (>—<) are considered undesirable as they are rarely the correct paths of the pen strokes which form these junctions. Matches within junction areas must be removed so that an alternative method, based on the vectors leading into the junctions, can be used.

The problem of bends and junctions is addressed by identifying the matches which surround the bend or junction areas and removing matches within those areas. Special nodes, called 'pixnodes' are created at the surrounding matches. Sections of parallel adjacent matches between pixnodes, and outside the junctions and bends, are vectorized by joining their midpoints. The remaining task is to decide how to join the pixnodes across the bend or junction areas. This is analogous to Pavlidis's compressed line adjacency graphs which consist of path nodes and junction nodes. The path nodes represent the easily vectorized sections and the junction nodes represent areas which require more complicated vectorizing. In the new method, sections of adjacent matches correspond to path nodes, and bends and junctions correspond to the arcs connected to junction nodes.

While the method can be used to create vectors with floating point components, a useful simplification is to round all points on the vector chain to integer coordinates. The loss of accuracy is very minimal (even with floating point the fractional parts of match midpoint coordinates would only be either 0 or 0.5) and the method is faster, simpler and produces vector chains with fewer points.

3.4. Pixel Edge Matching

It is assumed that the images are initially represented in outline form and are available to the vectorization process as linked structures of outline loops: chains of outer loops, each with subchains of any associated inner loops. The vectorization process operates on each outer loop (along with its inner loops) in turn.

The first stage in the process is to create a structure representing the edges of pixels along the outline. The outer loop and any inner loops it contains are traversed. Small outlines whose area is less than a constant, DOT_SIZE, may be ignored as noise. A doubly-linked list of structures, termed 'pixes', is created, with one 'pix' for every pixel edge traversed. The pix structure stores the (x,y) coordinates of the pix, pointers to the next and previous pixes in the list (*next* and *prev*), and a binary variable to indicate whether the pix is on a horizontal or vertical section of the outline (d), in addition to other information detailed later.

The coordinates of the pix are taken to be the coordinates of the midpoint of the pixel edge, with any fractional part rounded down to an integer. We wish to restrict coordinates to integers for speed of computation. However, this results in some horizontal and vertical pixes being mapped to the same point so a slightly complicated translation is required to keep all pixes at separate coordinates. We add 1 to the x coordinate of any pixes on horizontal sections of the outline, and add 1 to the y coordinate of any pixes on vertical sections of the outline.

Next, for the purpose of assigning array sizes, the following values are determined: the total number of pix structures created (*edge_cnt*), the minimum x and y coordinates of the outer loop (*x_min,y_min*), the width and height of the outer loop (*wth,hgt*).

An array, *pixpts[edge_cnt]*, of pointers to pixes is created. Each pix structure is pointed to by one element of *pixpts*. *pixpts* elements are indexed by the x,y coordinates of the pix using four other arrays:

col_cnt[wth]	- element n counts the number of horizontal pixes whose x coordinate is $n + x\_min + 1$

row_cnt[hgt]	- element *n* counts the number of vertical pixes whose y coordinate is *n* + *y_min* + 1
col_st[wth]	- element *n* stores the total number of horizontal pixes whose x coordinate is less than *n* + *x_min* + 1
row_st[hgt]	- element *n* stores the total number of horizontal pixes plus the total number of vertical pixes whose y coordinate is less than *n* + *y_min* + 1

Note that the '+1's are necessitated by the shift in coordinates going from outline to pix coordinates, described above.

These arrays are initially filled with zeroes. As illustrated in figure 3.2, the arrays *col_st* and *row_st* will provide indices to sections of the *pixpts* array corresponding to pixes in the same column or row of the outline.

Each pix indexes either the *col* or *row* arrays according to the relative coordinates of the pix to (*x_min*, *y_min*):

Index to *col_cnt* and *col_st* = *pix->x* - *x_min* - 1

Index to *row_cnt* and *row_st* = *pix->y* - *y_min* - 1

Initially the *col_cnt* and *row_cnt* elements indexed by each pix are incremented as the pixes are created. Once the counts are complete these two arrays are used to calculate the values of *col_st* and *row_st*.

The next stage is to step through the linked list of pixes and store a pointer to each pix in the *pixpts* array. Note that the values in *col_st* and *row_st* only indicate the starting index into *pixpts* of sections of pixes in the same column or row. Within each of those sections, a count must be kept of how many pix pointers have already been stored, in order to determine the exact index at which to store the next one. We reuse *col_cnt* and *row_cnt* for this purpose, after first resetting them all to zeroes again. Therefore, as the list of pixes is stepped through, the index at which to store a pointer to the pix in *pixpts* is determined as follows:

If the pix is on a horizontal section of the outline then the index to *pixpts* is found by summing the elements of *col_st* and *col_cnt* indexed by the pix. The *col_cnt* element is incremented.

If the pix is on a vertical section of the outline then the index to *pixpts* is found by summing the elements of *row_st* and *row_cnt* indexed by the pix. The *row_cnt* element is incremented.

Figure 3.2 illustrates the array formation for a simple outline. After *pixpts* is constructed the sub-divisions of pointers to horizontal pixes with the same x coordinate are sorted, within their sub-division, by their y coordinate. Similarly the pointers to vertical pixes with the same y coordinate are sorted by their x position. Matching pairs of pixes will now be adjacent to each other in the array.

Each pix structure contains a pointer (*match*) to its matching pix structure. These pointers are now set by stepping along *pixpts* two elements at a time, matching each adjacent pair of pix structures. Once all the pix structures have been paired, the matching process is completed. The task remaining is to decide which matches to keep for the purpose of forming vectors.

3.5. Match Quality

The basic quality score for a match is the width of that match in pixels. In certain cases the basic qualities will cause unsuitable matches to be kept or important matches to be deleted, e.g., on noisy edges, at splits and around holes in the image. Where these matches can be detected, the problems can be avoided by reducing or increasing the basic score.

In certain cases, described below, it is necessary to delete matches. This is most easily done by assigning the unwanted matches a high quality score which will guarantee their being removed at the match removal phase. Some maximum quality score must be determined for this purpose. For a given application, this is best determined from the maximum size of image producible by the scanner.

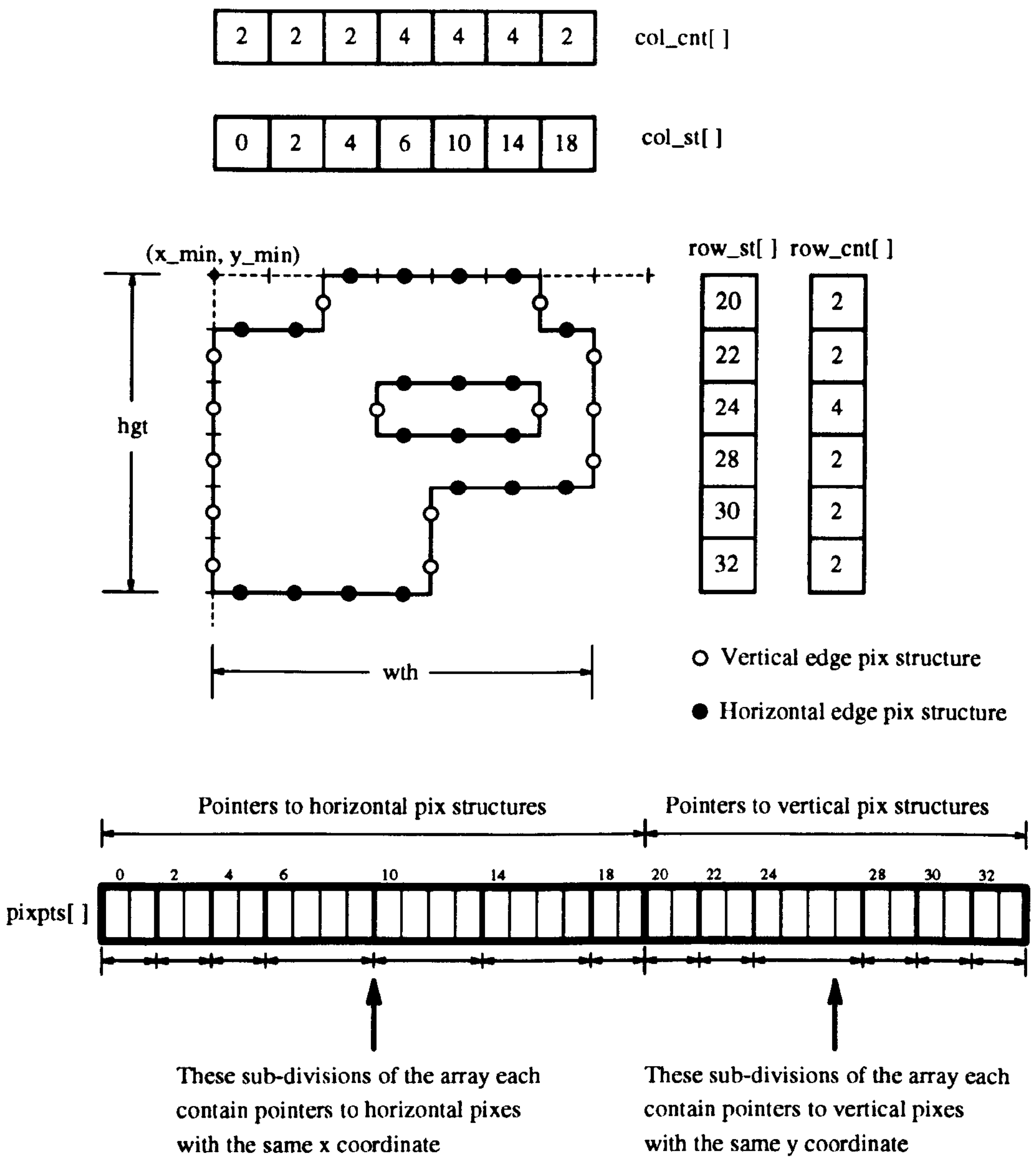

2 2 2 4 4 4 2
col_cnt[ ]
0 2 4 6 10 14 18
col_st[ ]
(x_min, y_min)
hgt
wth
row_st[ ] row_cnt[ ]
20 22 24 28 30 32
2 2 4 2 2 2
O Vertical edge pix structure
● Horizontal edge pix structure
Pointers to horizontal pix structures
Pointers to vertical pix structures
0 2 4 6 10 14 18 20 22 24 28 30 32
pixpts[ ]
These sub-divisions of the array each contain pointers to horizontal pixes with the same x coordinate
These sub-divisions of the array each contain pointers to vertical pixes with the same y coordinate

Figure 3.2 Formation of the *pixpts*, *col_st*, *col_cnt*, *row_st* and *row_cnt* arrays for a simple outline.

The quality score for a match is assigned to the quality elements (q) of the pixes at both ends of the match. This is done in two passes, one for the horizontal pixes and one for the vertical ones. For each outer loop, the first horizontal pix at the start of the linked list is recorded. A roving pix pointer, *rov*, then steps through the horizontal pixes in the list until it comes back to the start. Each pair of pixes (*rov*, *rov->match*) is assigned a quality score. The process is then repeated for the vertical pixes.

End zones and split zones, described below in sections 3.5.1 and 3.5.2, present special cases. To detect when these special cases occur a second pix pointer (called *vor*) is required. *vor* is set equal to *rov->match*. It is then advanced in the opposite direction to *rov*, i.e., when *rov* is advanced to *rov->next*, *vor* is advanced to *vor->prev*. Since we only wish to consider pixes in one direction, horizontal say, two more pix pointers are used: *probe*, to find the next horizontal pix forward from *rov*, and *eborp*, to find the previous horizontal pix backward from *vor*. (If we are working vertically then the pointers are to vertical pixes.) The following C code indicates how *probe* and *eborp* are advanced:

```
for (probe = rov->next; probe->d != dir; probe = probe->next);
for (eborp = vor->prev; eborp->d != dir; eborp = eborp->prev);
```

where *dir* is the direction under consideration.

The following sections describe the special cases of quality assignment and how the second pix pointer is used to determine when they should be applied.

3.5.1. End Zones

Short matches frequently occur on the edge of images, due to noise or spurs or simply the way a curved line is outlined. These matches would result in small spurs on the vectorized image running perpendicular to the true direction of the line. Also these short matches cause the correct ones which cross them to be deleted. This is illustrated in figure 3.3.

It is extremely rare for matches at the very edge of an outline to form part of the correct vectorization. Since they are at best useless and at worse erroneous, these matches should be removed. The question that remains is how deep in from the edges we should delete matches?

One method for removing edge matches, proposed by Cripps [Cri94] is to specify an *end zone* in which all matches will be deleted. The end zone starts at the match along the edge of the outline and works back from that edge through the adjacent co-directional matches until the criteria for zone membership is no longer met.

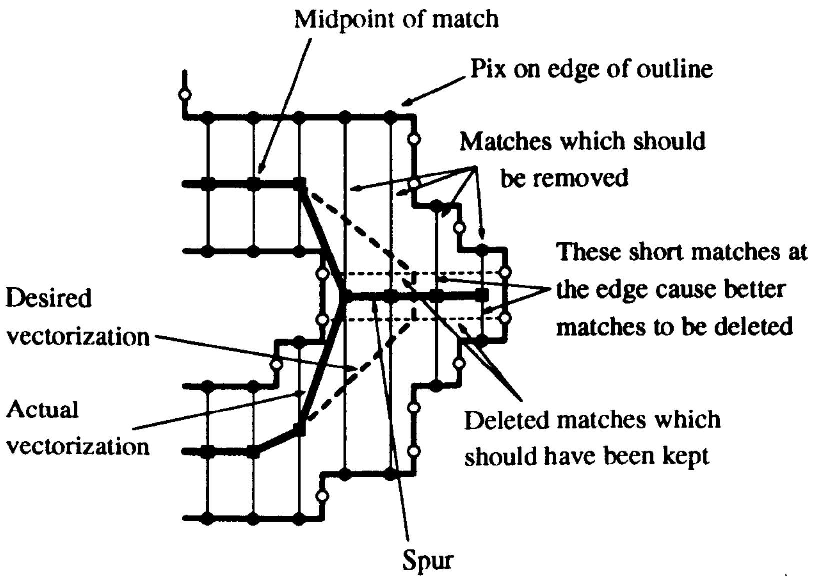

Midpoint of match
Pix on edge of outline
Matches which should be removed
These short matches at the edge cause better matches to be deleted
Desired vectorization
Actual vectorization
Deleted matches which should have been kept
Spur

Figure 3.3 Short matches along the edge of an outline can cause distortions and spurs in the vectorization.

Cripps' criteria is based on the width of matches in relation to their pixel distance from the edge. The match belongs to the end zone if:

i) The match width is greater than or equal to the previous match width (working away from the edge)

ii) The match width is greater than or equal to some constant factor, Q_END_FACTOR, of the pixel distance to the edge match

This method, which was designed for vectorization of lines on engineering drawings, proved unsuitable for character recognition. The long, thin lines of an engineering diagram are likely to cause the end zone to stop a short pixel distance from the edge. For the smaller, more detailed characters the zone can extend much further than desired, causing valuable matches to be deleted. Increasing the Q_END_FACTOR for character vectorization will improve the end zone specification. However, in many cases the match width increases in proportion to the distance for long stretches and the zone will still extend beyond what is necessary.

To prevent this, an additional criterion for end zone membership is required. An upper limit on the pixel distance from the edge is the simplest method; however, this limit will depend on the resolution and size of the image, and the amount of noise expected.

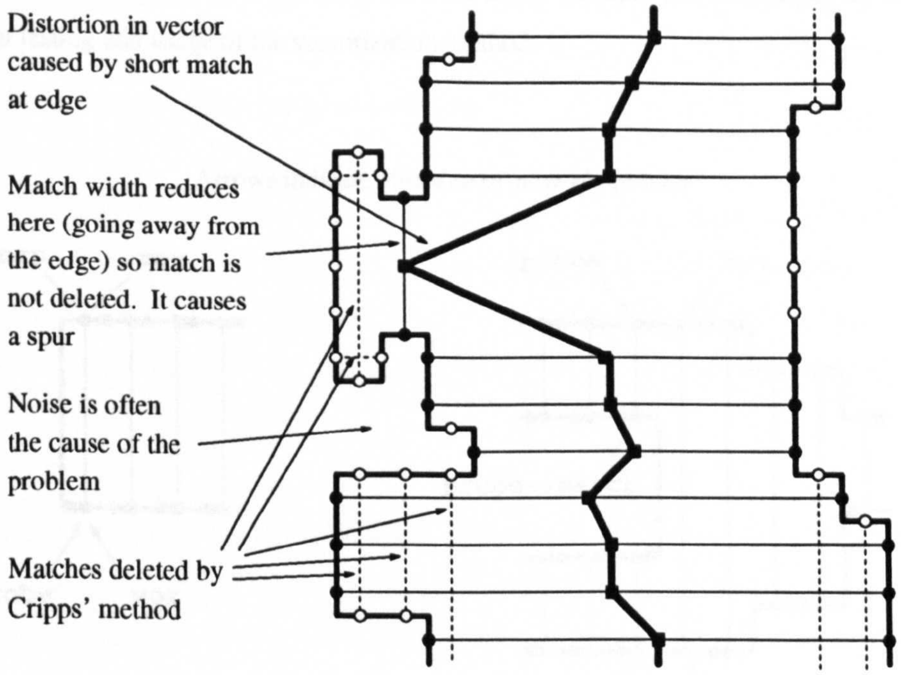

Distortion in vector caused by short match at edge
Match width reduces here (going away from the edge) so match is not deleted. It causes a spur
Noise is often the cause of the problem
Matches deleted by Cripps' method

Figure 3.4 A decrease in match width which should not signal the end of the end zone.

Another problem with Cripps' method is that a decrease in match width does not always indicate that it is safe to keep matches. Noise can cause situations where short matches close to the edge are not removed, resulting in distortions in the vectors, as shown in figure 3.4.

For character recognition, the Cripps' criteria for end zone definition are more complex than is really necessary. A simple removal of edge matches to a fixed depth limit is sufficient to eliminate the unwanted pairings. The precise depth limit will depend on the resolution, size and noise of the image representation and this dependency has not been explored in detail. However, it is typically much smaller than that allowed by Cripps' second criterion and is not difficult to determine by experimentation for a specific application. For the 32 x 32 pixel and 128 x 128 pixel, 300 pixels per inch character images used in this research, depths of between one and four pixels were tested. A depth of one pixel was insufficient as short edge matches occurred to a depth of two pixels in many cases. Depths of three and four pixels removed the short edge matches but were observed to remove useful matches in a number of cases. A fixed depth of two pixels very rarely allowed any undesirable short edge matches or deleted useful matches. It was therefore used as the standard end zone depth for the subsequent testing and usage of the vectorization method.

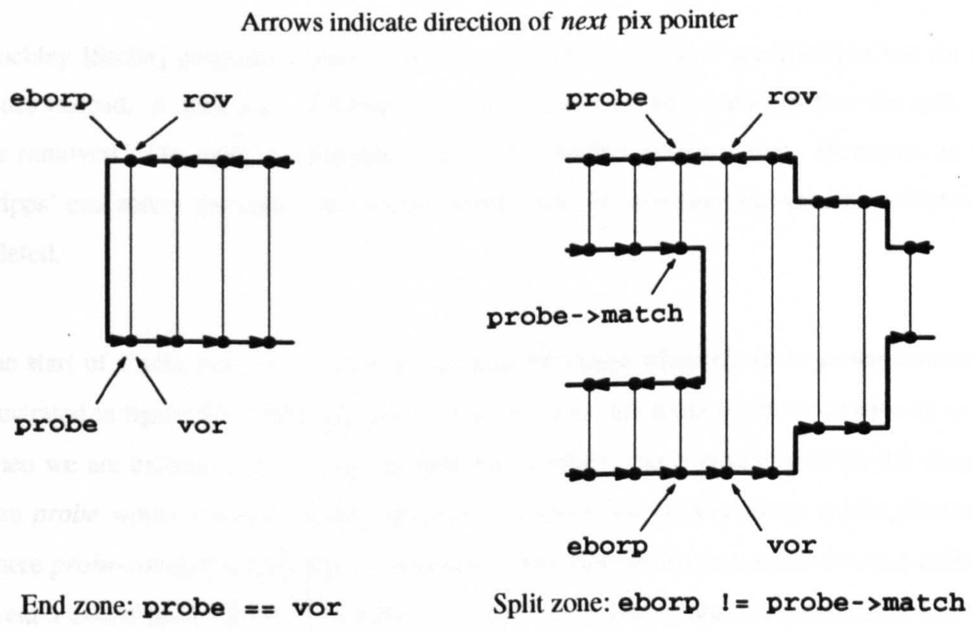

Arrows indicate direction of *next* pix pointer
eborp
rov
probe
vor
End zone: probe == vor
probe
rov
probe->match
eborp
vor
Split zone: eborp != probe->match

Figure 3.5 Conditions for detection of end and split zones. The diagram shows the positions of the pix pointers when *leaving* the split zone. The split zone condition is also true on *entering* the zone so an additional test is needed to determine whether the match (*rov* to *vor*) is inside or outside the zone.

The first match of an end zone is detected when *probe* = *vor*, as illustrated in figure 3.5. Subsequent matches in the same direction are also considered to be in the end zone. A count is maintained of the number of matches encountered so far in the zone. When the count exceeds the allowed depth, the zone has been left and subsequent matches are assigned quality normally (i.e., quality = match width).

3.5.2. Split Zones

Another case where matches cause problems are at splits in the outline. These are points on the outline where two or more image lines fork. As described in section 3.3, accurate junction vectorization requires the removal of matches from within junctions, and the identification of the matches which border those areas. Forks in the image occur near, or at, the borders of junction areas so quality assignment at these points must ensure the requirements for accurate junction vectorization are met.

Stockley [Sto94] proposes a similar method to Cripps' end zone specification but for split zones instead. A *split zone* is defined around the fork and all matches within the split zone are removed. The zone is intended to cover the desired junction area. However, as with Cripps' end zones, the split zones extend further than is necessary and valuable matches are deleted.

The start of a split zone is detected at a fork in the image when *eborp* != *probe->match*, as illustrated in figure 3.5. This condition is true when we are about to enter the zone as well as when we are exiting it, i.e., if we imagine *rov* is where *eborp* is indicated on the diagram, then *probe* would enter the zone, advancing to where *vor* is, and *eborp* would advance to where *probe->match* is indicated. Note that in this case *eborp* and *probe->match* have different x coordinates (when leaving the zone the x coordinates are the same). It is therefore simple to determine whether the match is inside or outside the split zone by checking whether or not the x coordinates of *eborp* and *probe->match* are equal for matches between horizontal pixes (for vertical pixes the y coordinates are checked).

Working round the outline, the split zone extends (and matches are removed) until one of the following conditions is met:

i) The pixel distance travelled from the start of the split zone exceeds a limit, Q_SPLIT_LIMIT

ii) The current match width is less than half the previous match width

iii) The match width has not decreased for a pixel distance of more than 2/3 of the current width

iv) A fork in the image has been reached

This can result in large areas being cleared of useful matches, particularly where, as is common, two or more split zones overlap. Consider the example presented in figure 3.6.

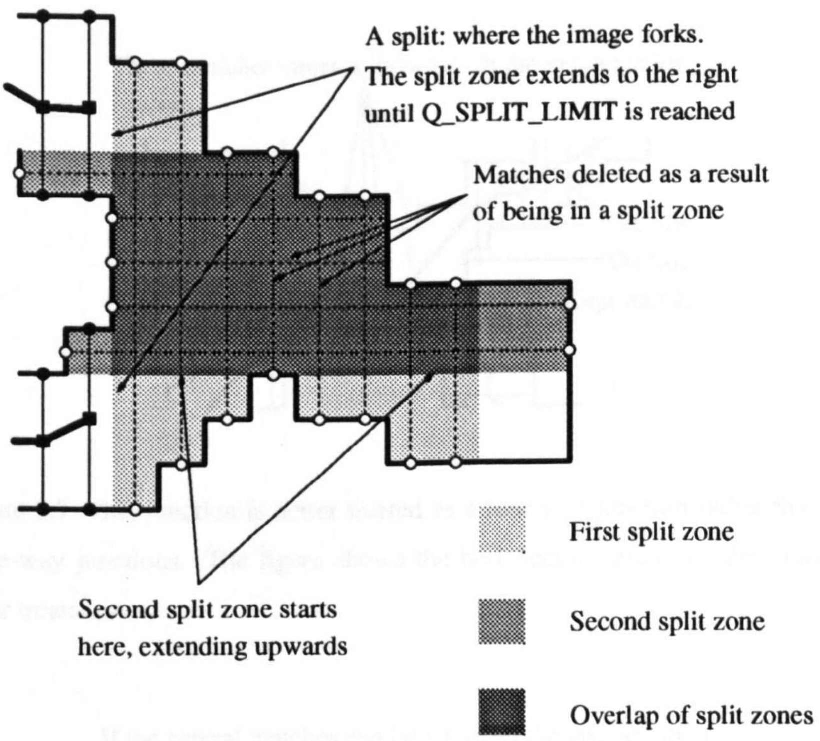

A split: where the image forks. The split zone extends to the right until Q_SPLIT_LIMIT is reached
Matches deleted as a result of being in a split zone
Second split zone starts here, extending upwards
First split zone
Second split zone
Overlap of split zones

Figure 3.6 The overlapping split zones cause a large region to be cleared of matches in both the vertical and horizontal directions. No matches are left to help the vectorization of the region.

Stockley's method will remove the short line on the right of the image. The method is designed for engineering drawing recognition where such short lines are regarded as spurs and are to be removed. In character recognition the line is an important detail which should be kept. It is therefore necessary to use a different approach which reduces the deletion of matches.

There is however a good reason for attempting to delete such large areas of matches. This is the problem of 'bow-ties.' The occurrence of these is a common problem with junction vectorization. Where matches are left in the middle of a large junction they effectively divide it into two or more smaller junctions. Vectorizing the smaller junctions separately is inferior to vectorizing them as a single large junction. A bow-tie-shaped (>–<) vectorization is produced as in figure 3.7. Figure 3.8 shows the desired vectorization, where the junction is treated as a single large junction.

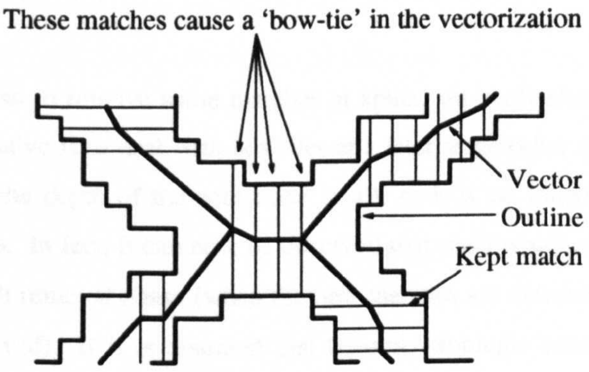


Figure 3.7 This junction is better treated as a four-way junction rather than two three-way junctions. The figure shows the best vectorization possible with the latter treatment.

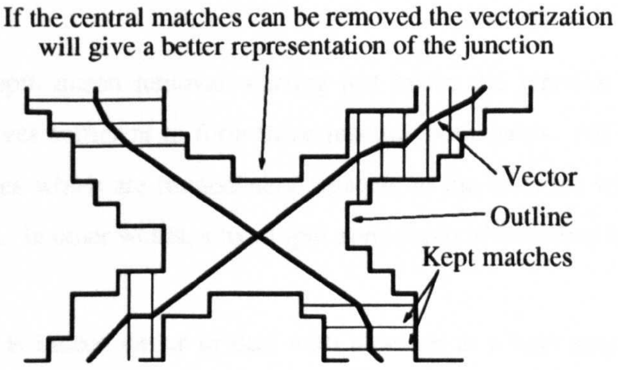

If the central matches can be removed the vectorization will give a better representation of the junction
Vector
Outline
Kept matches

Figure 3.8 Without the central matches the junction can be treated as a larger four-way junction. The vectorization is a better representation of the pen strokes.

It might therefore seem beneficial to remove as many matches as possible until we find another fork. Notice that in figure 3.8 the edges of the large junction are all at or near the forks. (If the match at the fork is crossed by a shorter one outside the junction then the junction boundary will be further along the fork, but at this stage it is not known which matches will remain so the match at the fork is the best available approximation of the desired junction boundary.) However, not every junction is such a large junction and forks are not the only desirable junction boundaries. Occasionally a bow-tie vectorization is correct (where the middle section is long or the angles of the forks are large) and the matches which make the middle bar should remain.

So although we wish to remove some matches at splits, we can't always remove them all. An effective alternative is to deal with bow-ties at a later stage (after match removal). This allows us to keep the depth of the split zone small, as it is no longer trying to cover the whole junction area. In fact, it can now be observed that most matches within junctions are deleted at the match removal phase (when the best matches are determined and those crossing them are removed). If it is assumed that bow-tie problems caused by the remaining matches can be dealt with later, then all that is really necessary is to ensure that junction boundaries are formed at the first shortest match along the forks. Often, as is the case in the above example, a line out of a junction in the image will cause forks in both the horizontal and vertical directions. There are effectively two boundaries to be created for the line, one for each direction. However, since the matches along the forks cross, only one shortest match will remain after match removal.

A simple fixed depth match removal, starting just inside the junction and working away from the fork, proves sufficient to form these junction boundaries. All that is needed is to remove the matches which are immediately adjacent to the ones we wish to keep and are inside the junction. In other words, a fixed split zone depth of one pixel is sufficient.

As it turns out, it is indeed easier to deal with bow-ties at a later stage, when we have a better idea of where the junctions are. We also have more idea of the bearings of lines entering the junctions. The simple method of deleting matches which immediately precede forks is therefore more appealing as it preserves the most stroke width information. It should also be noted that in tests on the NIST and CEDAR character data, described in sections 2.6.1 and 2.6.2, it was observed that Stockley's method did not always succeed in deleting matches across large junctions and an additional bow-tie removal process was required anyway.

3.5.3. Preserving Holes

One other case has been encountered in which match width does not work well as a measure of match quality. This is around inner loops in outlines, such as the holes in the character '8'. In some cases the holes are noise and should be ignored, but in many cases they are part of the character and the vectorization should circumscribe the hole. Using match width as match quality around these holes often fails to produce the desired vectorization.

The inner loops are small compared to the outer ones and have fewer matches around them to define their shape. Care must therefore be taken not to delete too many of the surrounding matches. If all the matches touching an inner loop on one side are deleted then there remains no accurate information about that side of the hole.

To help preserve the holes it is required to keep at least one of the matches from each of the four compass point-facing sides of the inner loop, unless it crosses a much better match. We cannot insist that these matches should always be kept as they may cross other such matches from different inner loops.

The proposed method is to find the longest match on each edge (north, south, east and west) of the inner loop. If it were the case that all but one match should be deleted from a side of the inner loop, then this match is generally the one that will provide the best definition of the hole. The quality of this match is set to half its width. This significantly reduces its chance of being deleted, without insisting that it will always be kept. The method is found to be extremely effective at ensuring that vectors are correctly drawn around holes.

3.6. Match Removal

Once quality has been set, it remains to sort the matches into ascending quality order and delete those which cross matches with lower quality scores. Profiling shows that this is the most time consuming process in the vectorization method [El193]. The ordering of matches in the *pixpts* array allows a binary chop to be used for speed.

Initially the *pixpts* array is ordered as in figure 3.9(a), with adjacent elements being matched together. From this arrangement, two orderings are produced within the array: the original ordering (horizontal pixes sorted in ascending x coordinate order, those at the same x coordinate further sorted by y coordinate; followed by vertical pixes in ascending y coordinate order, those at the same y coordinate further sorted by x coordinate); and ascending quality score order.

Since matched, adjacent pixes have the same quality and are linked by their *match* pointers to each other, all the original ordering information can be preserved using only one pix out of each pair. The first step in match removal is therefore to rearrange the *pixpts* array as follows:

> Let *half* equal half the length of the *pixpts* array;
>
> The odd numbered elements of *pixpts* are copied into a temporary array:
>
> > $temp[i] = pixpts[2i+1], \forall\, i: i \in \{0, 1, 2, 3, \ldots, half\text{-}1\}$;
>
> The even numbered elements of *pixpts* are copied into the element at half their index:
>
> > $pixpts[i] = pixpts[2i], \forall\, i: i \in \{0, 1, 2, 3, \ldots, half\text{-}1\}$;
>
> The temporary array elements are then copied back into the consecutive *pixpts* elements starting from halfway into the array:
>
> > $pixpts[i+half] = temp[i], \forall\, i: i \in \{0, 1, 2, 3, \ldots, half\text{-}1\}$.

The first half of the array now contains the first pix of each pair, preserved in the original order. The *col_cnt*, *col_st*, *row_cnt* and *row_st* arrays (which contain the indices in *pixpts* at which pixes, at given x or y coordinates, are stored) are adjusted for the new arrangement by simply dividing all their elements by two. The array is now ordered as in figure 3.9(b).

Next, the second half of the array is quicksorted [Hoa62] into ascending quality score order. The *pixpts* array has now been rearranged as illustrated by figure 3.9(c).

Next, the quality sorted matches in the second half of *pixpts* are worked through from the best (lowest scoring) to the worst (highest scoring). For each match (if it has not already been marked as deleted) it is required to find all matches which cross it and mark them as

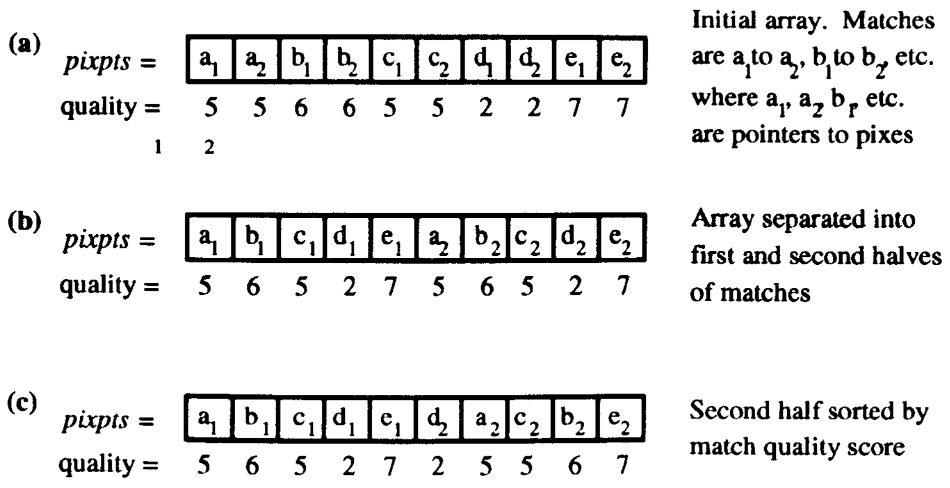

(a)
pixpts =
a1 a2 b1 b2 c1 c2 d1 d2 e1 e2
quality = 5 5 6 6 5 5 2 2 7 7
1 2
Initial array. Matches are a1 to a2, b1 to b2 etc. where a1, a2 b1, etc. are pointers to pixes
(b)
pixpts =
a1 b1 c1 d1 e1 a2 b2 c2 d2 e2
quality = 5 6 5 2 7 5 6 5 2 7
Array separated into first and second halves of matches
(c)
pixpts =
a1 b1 c1 d1 e1 d2 a2 c2 b2 e2
quality = 5 6 5 2 7 2 5 5 6 7
Second half sorted by match quality score

Figure 3.9 Arrangement of *pixpts* array before and after sorting, for a simple, ten element, example.

deleted. Note that there will be no crossing matches with lower quality scores as they will have been dealt with previously. Let *pp* denote the pix pointed to by the current *pixpts* element. The match, from *pp* to *pp->match*, is called the target match.

First the direction of the target match is determined. Vertical matches are indicated when *pp->d* (the direction of the pix at the end of the match) is horizontal; horizontal matches when *pp->d* is vertical. We will look at the algorithm for vertical matches. For horizontal matches the algorithm is the same, except that we have simply swapped x for y and row for column.

First recall the translation used to convert between outline coordinates and pix coordinates: for horizontal pixes, add one to the x coordinate; for vertical pixes, add one to the y coordinate (see section 3.4). This complicates the algorithm slightly as some of the coordinates used will not have been translated in the same direction as the values to which they will be compared.

Assuming a vertical target match, the only matches which might cross it are horizontal and are indexed in the first half of *pixpts* by *row_st* and *row_cnt*. The first step is to determine which rows to check, i.e., find the indices into *row_st* and *row_cnt* which correspond to the rows that need checking. The first and last indices are calculated as follows:

```
start = pp->y - ymin;
stop  = pp->match->y - ymin;
```

Note that these are 1 greater than normal as *pp*'s y coordinates haven't previously been translated. *start* should be less than *stop* so their values are checked and swapped if necessary.

The x coordinate of the target match is set:

```
target_coord = pp->x - 1;
```

This is decremented as it has previously been translated but it will be compared to pixes with untranslated x coordinates.

Now the *row_st* and *row_cnt* arrays are stepped through between the *start* and *stop* indices inclusive. The variables *lo* and *hi* are set to the indices in *pixpts* corresponding to the first and last pixes in the current row:

```
lo = row_st[i];
hi = lo + row_cnt[i] - 1;
```

where *i* is the index of the row being checked.

A binary search is used to move *lo* to the index of the pix with x coordinate closest to *target_coord* but not exceeding it:

```
while (lo < hi)
{
  mid = (lo + hi + 1) / 2;  /* mid = average of lo and hi, rounded up */
  if (pixpts[mid]->x <= target_coord)
    lo = mid;
  else
    hi = mid - 1;
}
```

The pixes in the first half of *pixpts* are always at the left (or upper) end of their match, so any such pix whose x coordinate exceeds *target_coord* cannot cross it. Since matches on a single row cannot overlap, any such pixes with x coordinates preceding *lo* cannot cross it either. It simply remains to check the match addressed by *pixpts[lo]* and mark it as deleted if it crosses the target:

```
if (pixpts[lo]->match != NULL &&        /* If the match hasn't been deleted */
    pixpts[lo]->x <= target_coord &&        /* and its ends are on either */
    pixpts[lo]->match->x >= target_coord)        /* side of target_coord */
  pixpts[lo]->match->match = pixpts[lo]->match = NULL;     /* Delete it */
```

This completes the processing for the row.

Once the match removal phase is complete, no matches will cross and only the ones with lowest scoring quality will remain. Matches which are only one pixel apart, and in the same direction, can now have their midpoints joined to form the correct vectorization of that part of the image. Where adjacent matches are more than one pixel apart or run in different directions, bend and junction areas result.

At the end of each of the initial vector chains, a special node called a 'pixnode' is created. Pixnodes mark end points and the borders of bends and junctions. Each pixnode is a structure storing a pointer to its associated vector chain. Other data is stored which is used to aid the bend and junction vectorization processes, including a state flag (to mark it as deleted, spliced, an end point, a junction point, etc.), the angle of the chain into a junction, the arity of the junction, and pointers to the other end of the vector chain, other pixnodes around the same junction and any other pixnode to which it must be spliced.

The remaining task is to determine a vectorization between pixnodes, across the junctions and bends, which best represent the image. Methods of vectorizing these more complicated areas are described in sections 3.7 and 3.8.

3.7. Bends

'Bends' are places where, after crossing matches are removed, two adjacent matches run in different directions. Figure 3.10 illustrates such an occurrence. Bends are so termed because they usually occur at bends in the image, however they can also occur on straight, sloping lines where the image does not actually bend (such as in the figure). A bend area is defined to be an unvectorized area (after the joining of adjacent match midpoints) which has exactly two bordering pixnodes.

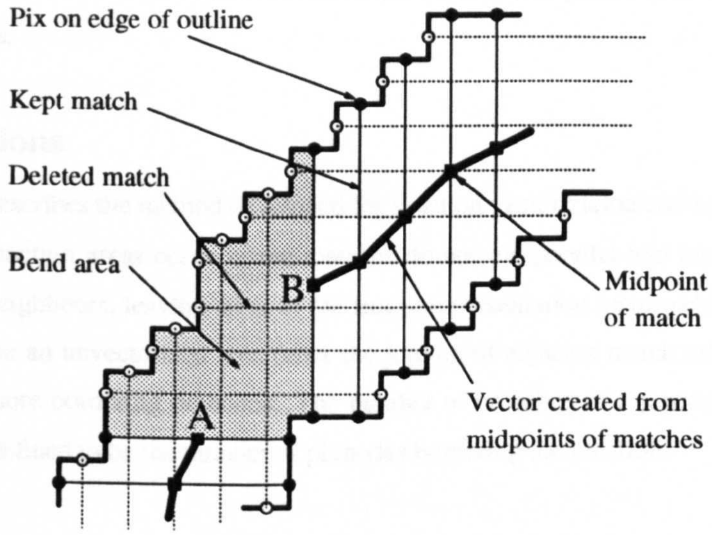


Figure 3.10 Example of a 'bend' area. In this case, simply joining pixnode A to pixnode B gives a reasonable approximation of the image line. In more general cases it is better to interpolate through the bend to give a more accurate vectorization.

Simply joining the midpoints of the matches which border the bend area does not always provide an accurate representation, particularly when the bend area is large. Instead it is preferable to interpolate the vector round the bend.

Interpolating round the bends is a simple matter of working along the edge pixes on either side of the bend simultaneously (sides are the outline borders of the bend area) from one bordering match to the other. This is similar to the methods used by contour tracking vectorization algorithms (see section 2.3.1.3). For each pix pair, a new vertex of the vector chain is added at the midpoint of the line between the pair. The process stops when the bordering match is reached on both sides. If the match is reached on one side before it is reached on the other the process continues but only the second pix is advanced. The vector chain around the bend is linked to the vector chains entering and leaving the bend at the appropriate points.

This rather crude method leaves many points on the chain. However, by rounding the mid-points to integer coordinates and deleting any duplicates the number of vectors can be kept to a minimum. If the vector chain is smoothed even fewer points will be necessary to describe bends.

3.8. Junctions

This section describes the method developed for junction vectorization and bow-tie removal. Like bends, junction areas occur where matches do not run parallel and immediately adjacent to their neighbours, leaving areas of the image representation unvectorized. A junction is defined to be an unvectorized area (after the joining of adjacent match midpoints) which has three or more bordering pixnodes. The number of branches of a junction is termed its 'arity' and is defined to be the number of pixnodes bordering the junction.

The final stage of the vectorization process is to determine a vectorization between the pixnodes, *across* the junction area, which best represents the image. Just what constitutes the best representation of a junction in an image depends on what is expected of the vectorized image. Some researchers lean towards making the vectorization an idealized abstraction of the image; others prefer the vectorization to represent the pen strokes made in drawing the character. This affects the way in which junctions are treated.

3.8.1. Bow-Tie Removal

Before junctions are vectorized the process of removing 'bow-ties' takes place. As discussed in section 3.5.2, bow-tie vectorizations occur where two three-way junctions occur close together. The short section of matches between the two junctions creates a short line joining them which is frequently unwanted. The junction should more properly be treated as a single, larger, four-way junction. Usually the pen strokes cross the junction and an 'X'-shaped vectorization should result. Occasionally, as described later in section 3.8.5, the pen strokes do not cross and an extra crossbar must be added (an 'H'-shaped vectorization). This crossbar does not necessarily run in the same direction as the bar of the bow-tie, however, so it is still better to remove the bow-tie. Note that the process is referred to as bow-tie removal but is more properly a bow-tie prevention process as only the crossbar is actually removed; the bow-tie itself is never allowed to form.

Section 3.5.2 described how matches immediately next to forks were removed so that junction boundaries would always occur at the forks. This ensured the four correct boundaries for the anticipated four-way junctions would be found. However, the process of removing other boundary matches within the junction (which cause bow-ties) was left to a later stage. That stage has now arrived.

The situation (after bends have been vectorized) is that only junction areas remain unvectorized. The rest of the image is now represented by vector chains. At each match bordering a junction there is a pixnode. From each pixnode a vector chain leads away from the junction, either to a dead end or to another pixnode at a junction boundary.

Bow-ties are avoided by removing short vector chains between three-way junctions. It is easier to do it at this stage as the vector chains between junctions allow a simple measure of the distance between them. Such a measure was not easily calculable before bends were processed. In addition to removing the chains, the matches which caused them must also be deleted to leave the desired larger junction. The larger junction can then be vectorized correctly.

A limit (called the BOW_TIE_LIMIT) on the length of connecting chains is set. Each pixnode is then stepped through in turn. The pixnode's vector chain, leading away from the junction, is stepped along and its length is incrementally calculated, stopping when one of the following is true:

i) Another pixnode is reached

ii) The end of the vector chain is reached

iii) The length exceeds BOW_TIE_LIMIT

If a pixnode is reached and borders a junction of arity three, and the length does not exceed BOW_TIE_LIMIT then the vector chain is stepped along once again. The matches corresponding to the points on the chain are removed. Finally the pixnodes on either end of the chain are marked for deletion. (They are not deleted until all the pixnodes in the image have been checked for bow-ties. This is so that the arity of junctions is preserved during checking.) After all the pixnodes have been tested, the marked ones are removed.

After this process, larger, four-way junctions should remain where desired. If two or more bow-ties are identified at a single three-way junction then the arity of the larger junction may be higher than four. Occasionally this can lead to larger junctions than intended. The problem then is to ensure the junction splicing method can vectorize these large junctions correctly. The subsequent sections detail the splicing process.

3.8.2. Methods of Dealing with the Odd Pixnode

In character recognition applications it can be safely assumed that the areas we have identified as junctions correspond to junctions of pen strokes, provided that short edge matches which might cause erroneous junctions have been removed (as described in section 3.5.1). Each pixnode of a junction will therefore be either an entrance or an exit of the pen stroke and the vectorization will pair up one entrance with one exit. Where a junction has odd arity there will be one pixnode left over.

If the vectorization is to be an idealized version of the character then this odd pixnode will usually be joined to a vector which crosses the junction. If it is to represent the pen strokes then it will be considered part of a stroke which terminates within the junction area. It will not be joined to crossing vectors but will be extended some distance into the junction and stop.

It will be seen later (in section 3.8.3.1) that the angle at which a stroke enters or leaves a junction can be roughly determined. One possibility for the extension of odd pixnodes is to extend the stroke from the pixnode at the angle of entry into the junction. This may be until it reaches the edge of the junction area, until it gets halfway across the junction, until it touches a crossing vector (in which case it may be joined to the crossing vector if so desired), or until some other distance is covered. Another alternative preferred by this research is to join the odd pixnode to the centre of gravity of the junction area. This attempts to accurately cover the junction area while approximating the pen stroke.

When the pen stroke terminates within a junction there is little information to indicate its precise path. The crossing pen strokes obscure it and the angle into the junction is only a rough approximation. Any of the suggested methods are therefore acceptable as none is generally more accurate than the rest.

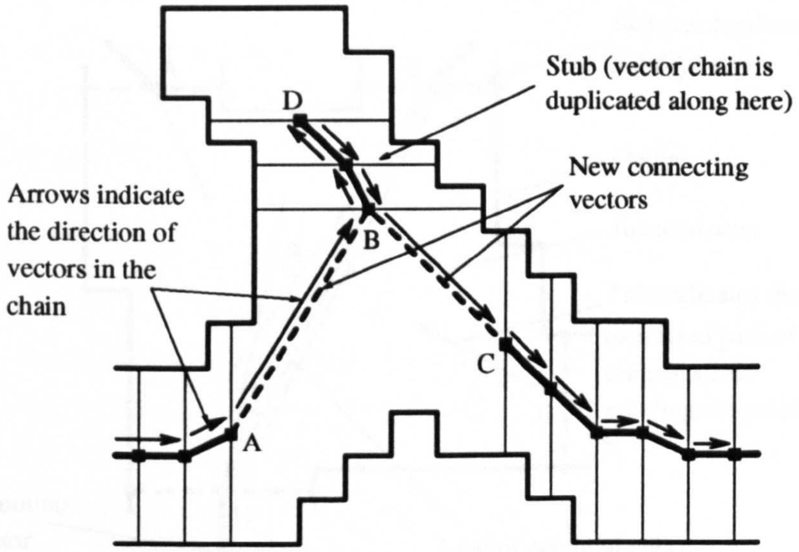
In the specific case of three-way junctions, where two pixnodes have been joined and one is left, a more complicated method of dealing with the odd node is used. A common cause of three-way junctions is a reversal of pen direction which overlaps the preceding stroke. The pen goes back on itself causing a three-way junction where one pixnode is both an entrance and an exit and should be joined to both of the other two pixnodes.

The length of the vector chain leading away from the odd pixnode is measured. If the chain terminates at a junction it is ignored. If it ends at a dead end and its length is shorter than a constant (called STUB_LENGTH) it is termed the 'stub' and is considered to be the result of overlap. STUB_LENGTH limits the allowed overlap.

The vectorization is then determined as illustrated in figure 3.11. The pixnodes A (the pixnode preceding the stub), B (the stub pixnode) and C (the pixnode succeeding the stub) are identified. The order of A and C is not important at this stage. The point, D, at the end of the stub is also found.

The existing splice across the junction (from A to C) is removed. New vectors are then added to the incoming chains. A is joined to B. The chain from B to D is followed, then an identical but reversed chain is added to lead back from D to B. B is joined to C.

The method is used after the rule of thumb for three-way junctions (section 3.8.4) has been applied but before quality-based splicing (section 3.8.3) is attempted. At this stage it is possible that no splice has been made across the junction. In this case the shortest vector chain which leads to a dead end and is shorter than STUB_LENGTH is taken to be the stub. The new vectors are then added as above. This is effective at vectorizing 'V' shapes. A three-way junction occurs at the bottom of the 'V' which most thinning and vectorization algorithms convert to a 'Y'-like shape. However, with the stub vectorization method, the duplicate vector chains of the stub will run down the point of the 'V', each joined on to one of the single chains leading up the prongs. After smoothing, it is likely that the duplicate chains will separate, moving into line with the prongs, thereby eliminating the 'Y'-like vectorization and giving the much more desirable 'V' shape.



Stub (vector chain is duplicated along here)
New connecting vectors
Arrows indicate the direction of vectors in the chain
A
B
C
D

Figure 3.11 Vectorization of a stub, where a pen stroke has gone back on itself causing a three-way junction.

Note that this is the only situation in which the method handles pen strokes going back on themselves. The inability of vectorization and thinning methods to deal with more complicated stroke overlaps has so far distinguished off-line handwritten character recognition from on-line recognition.

3.8.3. Splice Quality

A system has been devised for assigning a quality score to each of the possible pairings of pixnodes across a junction. The best quality pairings are joined, or *spliced*, by adding a vector from one to the other, joined on to the incoming vectors. The direction of some of the vectors along the stroke may need reversing so they are all in sequence. While there are two or more unspliced pixnodes, the joining continues based on the best splice quality of the remaining possible pairings.

A number of methods of assigning splice quality scores to pixnode pairings have been tested. Each pixnode is on the end of a vector chain which leads up to the junction. The splice qualities are all based on the angle at which these chains enter the junction.

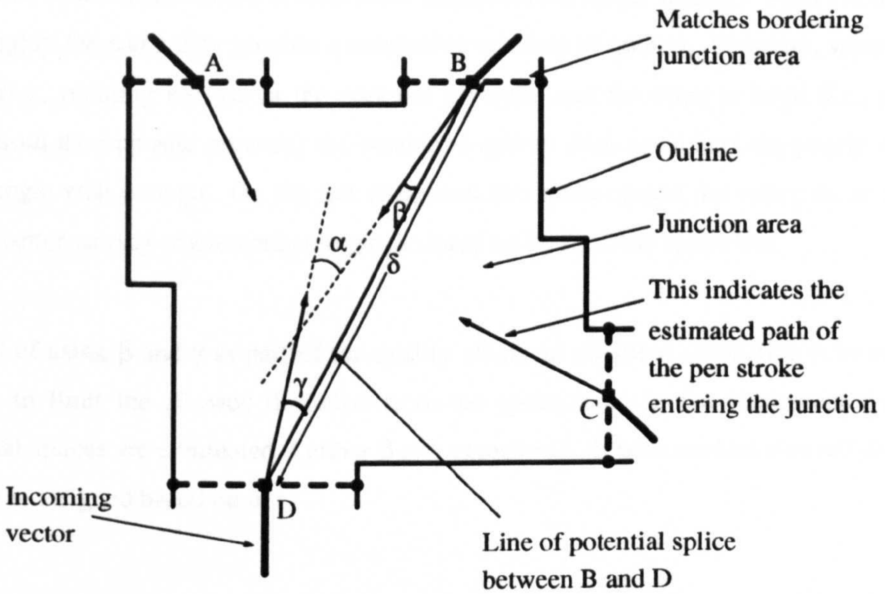

Matches bordering junction area
Outline
Junction area
This indicates the estimated path of the pen stroke entering the junction
A
B
C
D
α
β
γ
δ
Incoming vector
Line of potential splice between B and D

Figure 3.12 Angles considered during splice quality assessment for a potential splice between pixnodes B and D.

Figure 3.12 illustrates the angles used in determining splice quality for a specific potential pairing. Angle α, the angle between the two estimated lines of vector entry into the junction, ranging from 0 to π, is a good measure of how suitable a pixnode pairing is: the smaller the angle the better the splice. This is not always sufficient however. The angle alone contains no information about the position of the pixnodes relative to each other. Many pairings may have small values of α while the angles into the junction point away from the opposite pixnode.

The line directly between the two pixnodes, which is the line a potential splice would follow, serves as a guide to how closely the angles into the junctions point at the opposite pixnode. The angles β and γ (which are the angles between the estimated entry line into the junction and the potential splice line) are an additional indication of splice quality.

After observation of the splicing of test characters it was concluded that basing splice quality on angle α produced the desired result more often than the β and γ angles. The failure cases tended to be where β or γ was very large (greater than $\pi/2$). The poor performance of β and

γ was due to the combination of both these angles for the splice quality. Where both angles are roughly the same they provide a reasonable measure of quality. However, where one is small (i.e., pointing straight at the opposite pixnode) and the other is large (i.e., pointing away from the opposite pixnode) the combined quality does not reflect the poorly matched large angle well enough. On the test characters this often caused the splice to be selected over a better pairing where neither angle deviated so far from the splice line.

Instead of using β and γ as part of the quality value, an effective alternative is to use them simply to limit the allowed deviation from the splice line. In the implemented system, potential splices are eliminated if either β or γ exceed $\pi/2$. If both are less than $\pi/2$ the splice quality is assigned based on α.

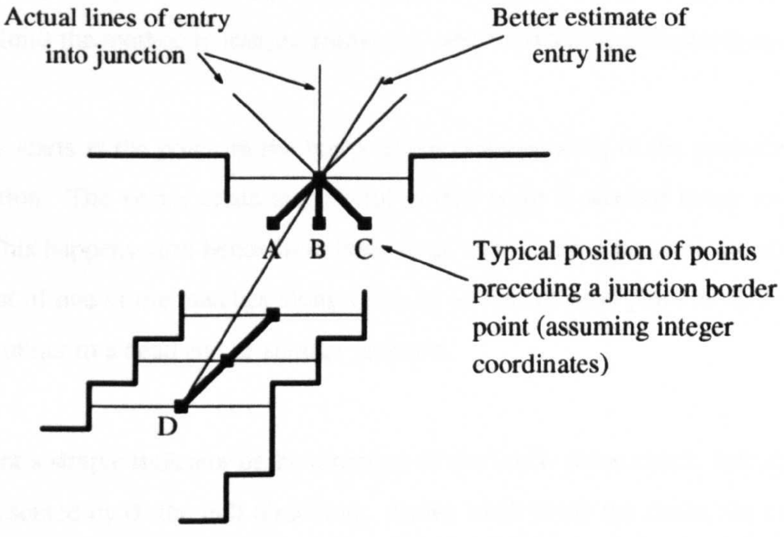
A further consideration is the distance across which the splice is to be made. Generally longer splices are preferred if they can be expected to be accurate, since short splices cover less of the junction. However, for long splices, the angle out of each pixnode points further away from the opposite pixnode than it would for shorter splices. Even for small values of α, a long match may be inferior to a shorter splice with a larger α angle.

One method attempted was to base the splice quality on the perpendicular distance from the entry line into the junction to the opposite pixnode. However, this distance measure does not account for the direction and degree of colinearity of entry lines and so quickly proved unsuitable. A simpler and more effective method is to use the distance between the two pixnodes (the length of the potential splice line), marked in figure 3.12 by δ, to scale the splice quality. This allows close pixnodes to be spliced more easily but will usually favour long splices where the intersection of the opposing entry lines is close to head-on.

The final choice for splice quality calculation used in this research therefore, is to eliminate splices from consideration if β or γ exceeds $\pi/2$, and otherwise to set quality to $\alpha\delta$. Low scores indicate better quality. On a sample of two hundred images requiring splices, taken from the NIST and CEDAR segmented character databases, the desired splices were obtained in 96% of the cases. (This includes the special cases of three-way and four-way junction splicing described in sections 3.8.4 and 3.8.5.)

3.8.3.1. Approximating Entry Angle into Junctions

Since integer coordinates are envisaged for use with this method, the actual angle of entry into the junctions of the vector chain is not a useful measure. Consider figure 3.13. Typically the entry path will be either A, B or C, in this case giving entry angles of $\pi/4$, 0 or $-\pi/4$ respectively. Since the approximation of pen stroke entry is vital to the success of the splicing method, much greater accuracy is required. It is therefore necessary to find a point further back along the incoming vector chain. The line from this point through the actual entry point (the midpoint of the match bordering the junction) is then taken to be an estimate of the true line of entry of the pen stroke, illustrated by line D in the diagram.



Actual lines of entry into junction
Better estimate of entry line
A
B
C
Typical position of points preceding a junction border point (assuming integer coordinates)
D

Figure 3.13 Typical actual entry lines of vector chains into junctions (assuming integer coordinates are used). The line from point D is a better estimate of the true path of the pen stroke.

(Note that the terms 'incoming' and 'entry' are used to refer to a vector chain leading to or from a junction. These terms have no relation to the actual directions of the vectors, which at this stage are meaningless.)

One method is to work back a fixed distance along the incoming vector. This however does not take into account the possible turning of the vector along the fixed distance. Often the vector will bend around considerably and the resulting angle is very different from the true angle of entry.

Determining a fixed distance which would be long enough to give a good approximation of entry angle, while being short enough not to go around bends, proved an impossible compromise. A similar method which works back a fixed number of points on the vector chain has the same problem. It is necessary to consider the change in angle as we work back and use this to indicate when to stop.

We have developed a method which works back along the incoming vector, provided it doesn't turn outside of a certain tolerance, or change the direction of its turning. An approximation of the turning is used, which makes use of the nature of the vector chains along line sections to limit the method to integer arithmetic, while remaining acceptably accurate.

The process starts at the point on the vector chain corresponding to the pixnode at the edge of the junction. The vector chain leading up to that point is worked along away from the junction. This happens after bends have been dealt with, so each point on the chain is either the midpoint of one of the matches along a line or an interpolated point around a bend. The chain leads either to a dead end or another junction.

At each point a simple indicator of the direction of the line is determined. Initially the direction is represented by 0 (the null direction). As we work along the chain, the current direction is determined by the coordinate change from the previous point on the chain. Let (x_C, y_C) be the coordinates of the current point and (x_P, y_P) be those of the previous point. The direction indicator is set as follows:

When we are working up or down a vertical section of the chain:

```
if ($x_C > x_P$)        /* current point is to the right of the previous point */
  direction indicator = 1;
else if ($x_C < x_P$)   /* current point is to the left of the previous point */
  direction indicator = -1;
else if ($x_C == x_P$)
  direction indicator = 0;
```

When we are working left or right along a horizontal section of the chain:

```
if ($y_C > y_P$)            /* current point is below the previous point */
  direction indicator = 1;
else if ($y_C < y_P$)       /* current point is above the previous point */
  direction indicator = -1;
else if ($y_C == y_P$)
  direction indicator = 0;
```

By the direction, horizontal or vertical, of the section of the vector chain we mean the direction of that chain at the edge of the junction — the opposite direction to that of the match bordering the junction.

This provides a rough indication of the direction of the line, which can be used to identify when the direction of turning changes. Two direction indicators are kept, one for the current point and one for the overall direction of the chain section along which we will work back. The latter (termed the *overall direction indicator*) is initially 0 and can only be set to 1 or -1 once. This happens the first time the current direction indicator is set to a non-zero value. Once set, the overall direction indicator becomes the fixed direction of the line into the junction.

Note that the approximation works because the incoming chain cannot turn too sharply, even in the direction of the overall indicator, without going round a bend. As explained below, the process always stops if it reaches a pixnode point. Such points occur at the bordering matches of bends so the method will stop before any sharp turns are traversed.

The vector chain is stepped along, point by point, starting at the edge of the junction and working away from it. A count is kept of the number of points traversed. The process stops if one of the following conditions is met:

i) The overall direction indicator is still 0 (unset) after a fixed number of vector points, FIRST_NULL_LIMIT;

ii) A pixnode is reached, other than the one at which we started;

iii) The current direction indicator is different from the overall direction indicator except where either:

a) The overall direction indicator is 0 in which case it is set equal to the current indicator;

b) The current direction indicator is 0 and no more than a fixed limit of vector points, SECOND_NULL_LIMIT, have been traversed.

If, after the process stops, the current and overall direction indicators are different then we have gone one point too far, so we back up one point along the chain. In some cases the current point finishes up at the same coordinates as the point at the edge of the junction (they may actually be the same point). This frequently happens when a bend occurs immediately adjacent to a junction (the process stops after the first step round the bend and backs up to where it started). In this case we move the current point along the chain until it reaches a pixnode other than the one at the edge of the junction — this will be the pixnode at the other side of the bend. (Note that it is acceptable to traverse a bend which occurs immediately adjacent to the junction as it is our only indication of the incoming path; otherwise we should stop at the near edge of the bend to avoid sharp turns.) Occasionally, however, this occurs when there is only one match along the stroke leading into the junction, the match that borders it, and so the current point can never progress further than the start point.

At this stage, the line between the current point and the point at the edge of the junction gives the approximation of the line into the junction. The bearing of this line is recorded as the entry angle into the junction for that incoming vector. In the cases where the current point is still at the start point, the line into the junction is assumed to be perpendicular to the match at the junction edge. The angle into the junction is determined accordingly.

The constants, FIRST_NULL_LIMIT and SECOND_NULL_LIMIT, will depend on the size of the image representation. For the NIST and CEDAR alphanumeric characters the values 4 and 5 were used respectively.

This method gives a fair approximation of the entry angle and is clearly less error-prone than the blind, fixed-distance backtracking method which was first attempted. In combination with the splicing algorithm, highly accurate splicing results have been produced (see section 3.8.3).

3.8.4. Three-Way Junctions

It was observed that for three-way junctions where, of the three bordering matches, two were horizontal and one was vertical, or vice versa, the correct splicing was (in the large majority of cases) to join the vector chains of the two co-directional matches. This simple heuristic is used whenever possible. In cases where all the bordering matches are in the same direction, the normal splice selection method is used.

After this rule of thumb has been applied but before the normal splicing method is used, the stub vectorization process described in section 3.8.2 is used.

3.8.5. Four-Way Junctions

A special case is also made for four-way junctions. We simplify the pairing possibilities by assuming that each of the two incoming strokes must lead to separate outgoing strokes. This is true in almost all cases.

Initially a simple rule of splicing pixnodes alternately was tried. This is roughly equivalent to Pavlidis' compound vectorization where a junction node is bordered by four intervals. This was found to produce imperfect results in many cases, particularly after bow-tie removal (see section 3.8.1), where the pen strokes at the junction have not actually crossed. In these cases two pen strokes have passed between adjacent pixnodes (adjacent round the junction). Usually a further pen stroke forms a crossbar between the other two strokes; no evidence remains of this, it having been incorrectly removed during bow-tie removal.

Consider a capital 'H'. Three-way junctions occur at either side of the crossbar, joined by a short vector chain representing the bar. As described in section 3.8.1, the bow-tie removal process will sometimes (incorrectly) remove the chain representing the bar to leave a larger, four-way junction where the crossbar should be. A similar character is shown in figure 3.14. In this case splicing pixnodes alternately (joining pixnodes A to C and B to D), producing an 'X'-like crossing of the junction, is incorrect.

Splicing of adjacent pixnodes (A to B and C to D) should therefore be allowed but an additional vector chain, forming the crossbar, must also be created in these cases. This crossbar vector can be interpolated between the two splices in a similar way to that in which bend points are interpolated. We step along the edge pixes on either side of the junction from the ends of one splice to the ends of the next; at each step we add the midpoint of the line joining the two pixes to the vector chain. More simply it can be a single straight line joining the midpoints of the two splices (as shown by the dotted line in figure 3.14).

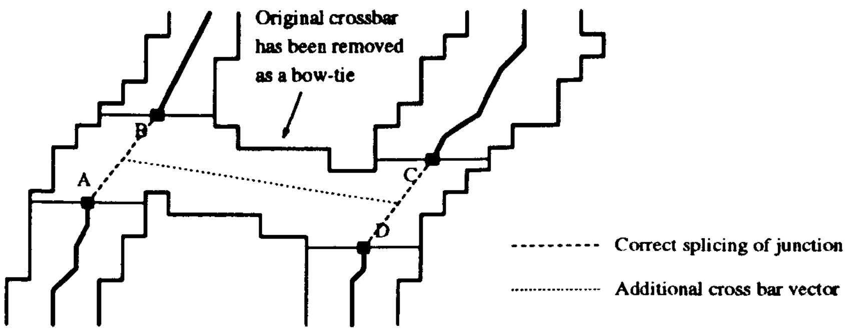

Original crossbar has been removed as a bow-tie
A
B
C
D
Correct splicing of junction
Additional cross bar vector

Figure 3.14 A four-way junction where alternate splicing of pixnodes (A-C and B-D) is inappropriate. Adjacent splicing (A-B and C-D) is correct but an additional crossbar vector is needed to accurately represent the junction.

As for higher arity junctions, splice quality measures are used to determine which pixnodes to splice. However, since making one splice automatically determines the second splice (e.g., if the first splice is A-B the second must be C-D) greater accuracy is achieved by adding the qualities of the two splices to produce an overall quality for each possible splice combination. (Multiplying the qualities was also tried but adding was found to work best.) This is a significant improvement which helps avoid choosing combinations where the first splice has very good quality but the second is poor.

In theory this quality combination could, at greater computational expense, be extended to six, eight or higher arity junctions.

3.9. Additional Preprocessing

Additional preprocessing may be applied to vectors to obtain a smoother form, to mend breaks in the image or to further normalize the vector representation.

Smoothing is achieved by analysis of each run of three consecutive coordinate points on the vector chain. The perpendicular distance of the second point to the line joining the first and third point is calculated. If it falls within a fixed tolerance the second point and the two vectors joining it are deleted and a new vector is created from the first point to the third point. This removes a large number of excess points, in particular duplicate points that may be created in the bend vectorization.

Breaks in the outline are repaired after vectorization by joining the ends of vector chains which fall within a fixed distance. The ends are joined by adding a new vector from one end to the other and reversing the direction of one of the initial chains if necessary.

Other forms of preprocessing, such as skew and size normalization, can be applied using similar methods to those for pixel or outline image representations. Figure 3.15 illustrates the effectiveness of smoothing in reducing the number of points on the vector chains and improving the appearance of the strokes. The ability to repair breaks in images by vector chain end point joining is also demonstrated.

3.10. Conclusions

A vectorization method has been adapted and developed from the algorithms of Pavlidis and Elliman. While it is slower than Pavlidis's method, it gives a much more accurate representation of characters, particularly at junctions where pen strokes cross. The basics of the method have already been proved to be effective on engineering drawings [Cri94] [Sto94]. In this chapter it has been improved, developed and extended to suit the vectorization requirements of handwritten characters. Although it is difficult to compare different vectorization methods, the ability of this method to preserve character shape and accurately vectorize crossing pen strokes can be seen from figure 3.16.

The encoding of strokes as vector chains provides a more compact representation of characters and may allow high level features to be extracted for improved accuracy (the use of strokes as primitives might be effective, provided that strokes can be accurately extracted in the presence of noise and breaks). Future developments of the method would aim at more detailed detection and vectorization of overlapping pen strokes, and at determining the time ordering and direction of strokes, with a view to applying on-line recognition techniques to

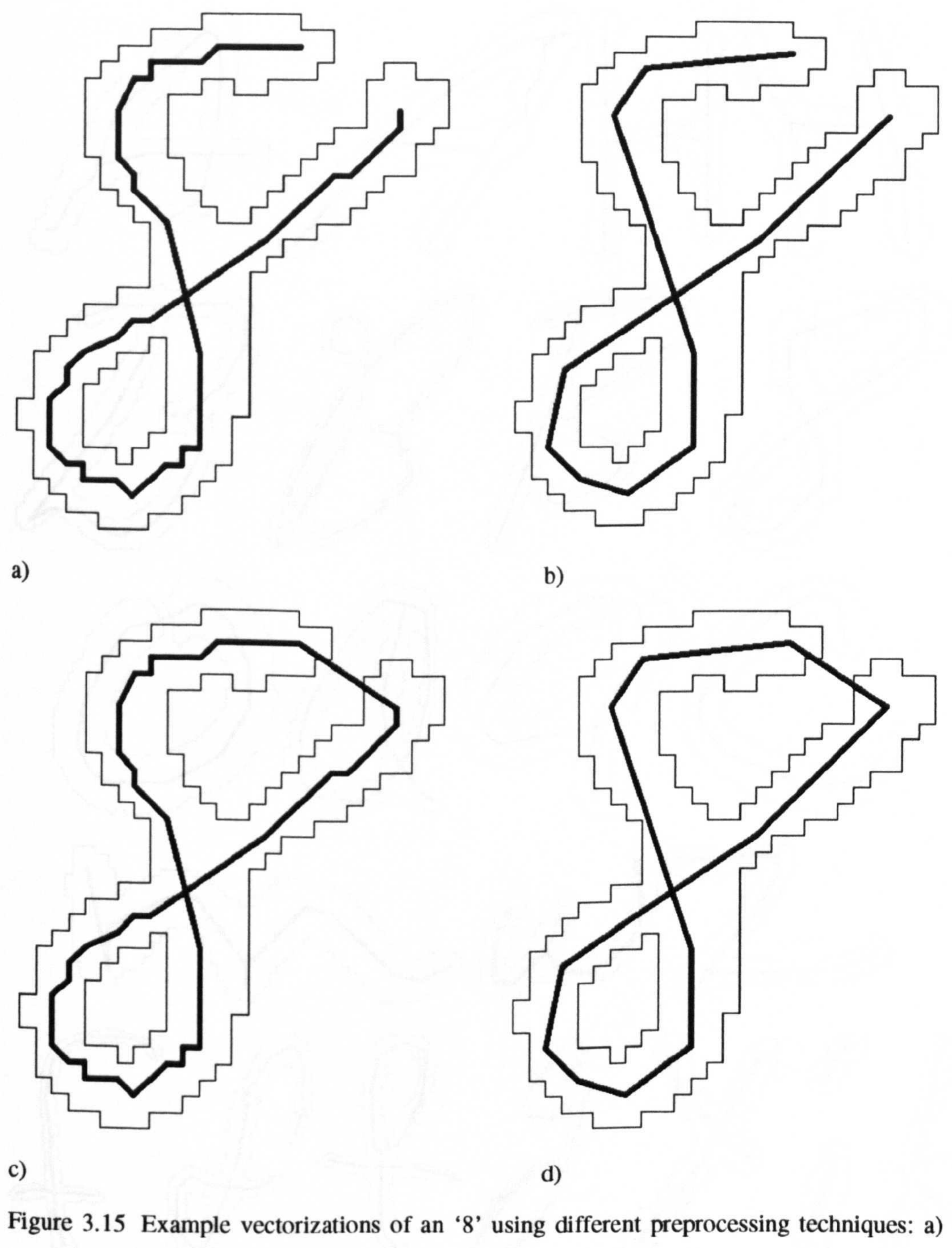


Figure 3.15 Example vectorizations of an '8' using different preprocessing techniques: a) with no preprocessing; b) with smoothing within a tolerance of 1.0 pixel widths; c) with end point joining within 8.0 pixel widths; d) with both smoothing (within a tolerance of 1.0 pixel widths) and end point joining (within 8.0 pixels). The thin lines are the outlines; the thick lines are the corresponding vector chains.

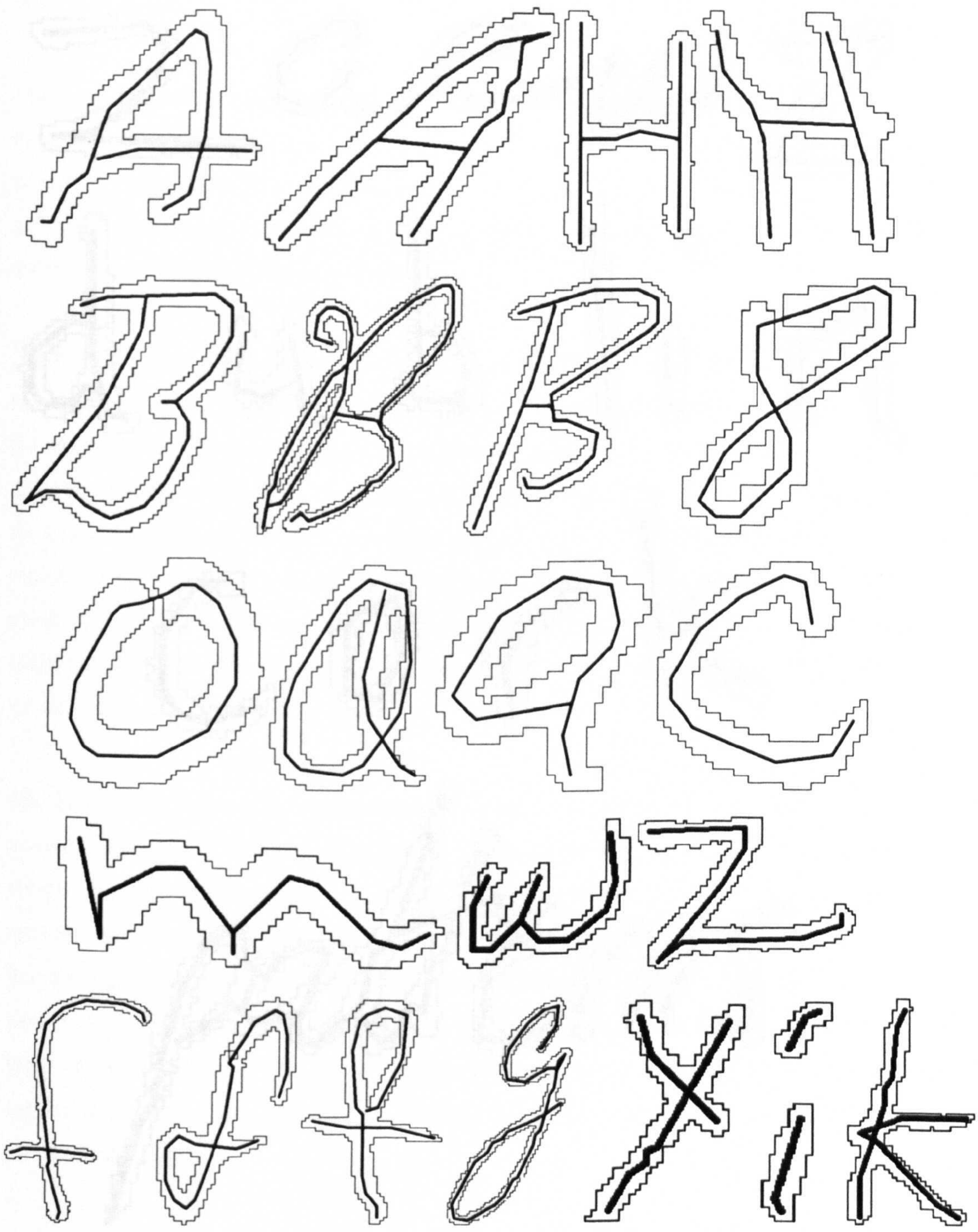


Figure 3.16 Example vectorizations from the NIST and CEDAR databases. Smoothing and end point joining have been used. The thin lines are the outlines; the thick lines are the corresponding vector chains.

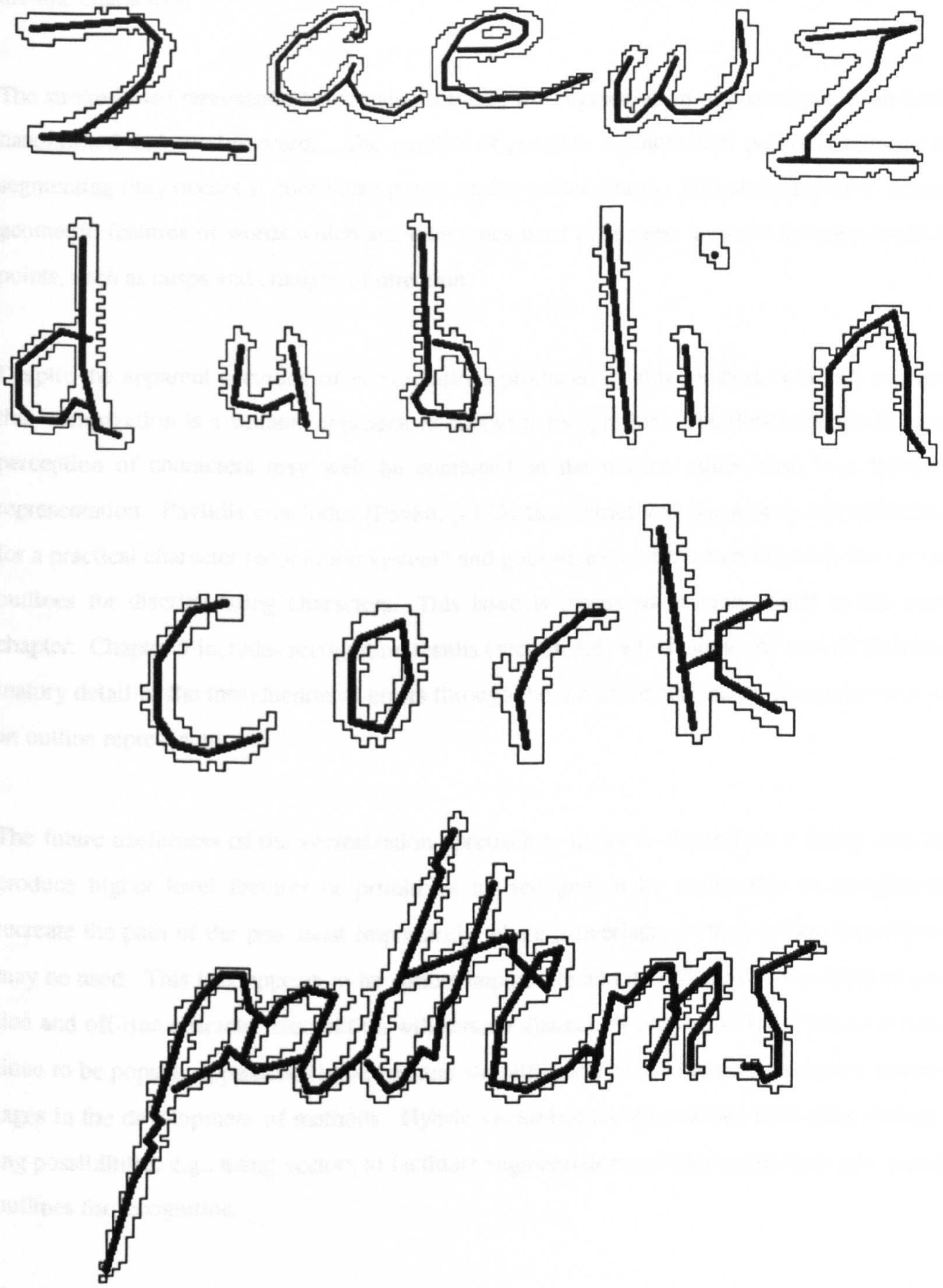


Figure 3.16 continued. Note the difficulty of vectorizing cursive script where pen strokes frequently overlap.

off-line characters.

The stroke-based representation can also simplify the segmentation of characters, from both handprinted and cursive words. The number of possible segmentation points is reduced if segmenting only occurs at coordinate points on the vector chains. It is also simpler to detect geometric features of words which are sometimes used to suggest appropriate segmentation points, such as cusps and changes of direction.

Despite the apparent accuracy of vectorizations produced by this method, it is still unclear that vectorization is a suitable approach to character recognition. The detail used in human perception of characters may well be contained in the outline rather than in a thinned representation. Pavlidis concludes [Pav86, p.125] that "strict vectorization is not sufficient for a practical character recognition system" and goes on to say he is investigating the use of outlines for discriminating characters. This issue is discussed in more detail in the next chapter. Chapter 5 includes recognition results (section 5.6) which show the loss of discriminatory detail or the introduction of errors through the use of vectorization, in comparison to an outline representation.

The future usefulness of the vectorization approach is likely to depend on it being able to produce higher level features or primitives for recognition by being able to completely recreate the path of the pen, most importantly where it overlaps, so that on-line techniques may be used. This task appears to be almost impossible and it is likely that the fields of on-line and off-line character recognition will remain distinct. Vectorization and thinning continue to be popular approaches however, and simplification of the image offers many advantages in the development of methods. Hybrid vector/outline approaches also offer interesting possibilities, e.g., using vectors to facilitate segmentation and then using their associated outlines for recognition.

3.11. Nomenclature

$\forall$	For all.
$\in$	Element of.

α	Angle between two estimated lines of vector entry into a junction.
β, γ	Angles between the estimated lines of vector entry into a junction and the potential splice line.
δ	Length of a potential splice line.
BOW_TIE_LIMIT	Constant which limits the length of the crossbar of a bow-tie.
col_cnt	Array of number of horizontal pixes with a given x coordinate.
col_st	Array of total number of horizontal pixes left of a given x coordinate.
d	Horizontal or vertical direction indicator in pix structure.
dir	Direction (horizontal/vertical) of pixes currently under consideration.
DOT_SIZE	Constant area below which outlines are considered noise.
eborp	Variable pointer to a pix structure that stays one step behind *vor.*
edge_cnt	Total number of pix structures created.
FIRST_NULL-_LIMIT	Constant which limits the backtracking along a vertical or horizontal vector when estimating entry angles into junctions.
half	Half the length of the *pixpts* array.
hi	Index into *pixpts* array of the last pix in the current row.
hgt	Height of the outer loop.
lo	Index into *pixpts* array of the first pix in the current row.
match	Pointer from pix structure to its matching pix structure.
pixpts	Array of pointers to each pix structure of a character.
pp	Pix pointed to by the current *pixpts* element.
probe	Variable pointer to a pix structure that stays one step ahead of *rov.*
q	Element of pix structure that stores the associated match quality.
Q_END_FACTOR	Constant which limits the size of an end zone.
Q_SPLIT_LIMIT	Constant which limits the size of a split zone.
rov	Variable pointer to a pix that moves forwards through linked list.
row_cnt	Array of number of vertical pixes with a given y coordinate.
row_st	Array of total number of vertical pixes above a given y coordinate.

SECOND_NULL-_LIMIT	Constant which limits the backtracking along a vector in the same direction as the overall direction indicator or in the null direction when estimating entry angles into junctions.
start	Index into *st* and *cnt* arrays.
stop	Index into *st* and *cnt* arrays.
STUB_LENGTH	Constant which limits the length of stub chains.
target_coord	Coordinate of target match.
vor	Variable pointer to a pix that moves backwards through linked list.
wth	Width of the outer loop.
x_min	Minimum x coordinate of outer loop.
y_min	Minimum y coordinate of outer loop.

Chapter 4

The Preprocessing Approach — An Appraisal

4.1. Summary

This chapter discusses the traditional approach to character recognition, which performs a number of preprocessing operations on character images before feature extraction and classification. Section 4.3 examines the usage and effectiveness of the various preprocessing techniques. Section 4.4 presents an alternative approach to the problem which attempts to avoid preprocessing as much as possible. The main reasons for questioning the use of preprocessing are introduced in section 4.2 and the specific problems of particular techniques are discussed in more detail in section 4.3.

The purpose of this chapter is to explain the minimal preprocessing approach which has been taken in the remainder of the work presented in this thesis. It suggests that some preprocessing techniques are detrimental to the accuracy and generalization of recognition. It also suggests that many of the useful strategies can be implemented in more effective and efficient ways during the later stages of recognition. However, a few techniques, such as slant correction, are still best performed as preprocessing operations.

The large number of different preprocessing methods makes a quantitative comparison beyond the scope of this thesis. This chapter is simply a qualitative discussion of the approaches.

4.2. The Preprocessing Approach

Preprocessing, as referred to in this chapter, is the processing of an image representation prior to presentation to the feature extraction stage of character recognition (or prior to the classification stage in systems which do not have a conventional feature extraction stage). This processing is a complete alteration of the image representation, by which it is meant that an operation is performed on the entire original representation to produce an entirely new representation (usually in the same format) to replace the original.

Preprocessing is almost universal in character recognition. It has become the standard approach, to scan the document, perform a selection of preprocessing operations on it and then to extract features and classify characters. Some variants of this approach exist. For example, recursion may be introduced to perform corrections of earlier stages, or the feature extraction stage may be bypassed, leaving the classifier to recognize the image representation directly. However, almost all systems perform some form of preprocessing on the raw image representation.

The various types of preprocessing techniques used have been surveyed in section 2.3. Briefly, these are thinning and vectorization, polygonal approximation, smoothing, filling and joining, and normalization. The purpose of this chapter is to examine the usage and effectiveness of these preprocessing techniques and to discuss their validity in the context of the whole recognition system. Chapter 3 has already raised doubts about the worth of thinning and vectorization. This questioning is now continued and extended to other types of preprocessing.

The purpose of this chapter is only to discuss preprocessing. It does not attempt to present quantitative results or prove the validity or otherwise of this approach and is included in this thesis mainly to represent the author's thoughts on the matter of preprocessing and serve as an explanation for the alternative approach taken in future chapters. Nonetheless, it does present some qualitative observations of the weaknesses and limitations of certain preprocessing techniques that suggest they are unsuitable for use in a character recognition system.

The intention of preprocessing is to make characters of the same class more alike by eliminating differences between them that are not relevant to their recognition. The main reason for doubting the efficacy of preprocessing techniques is that almost all researchers approach the problem with preconceived ideas about what characteristics are or are not relevant to recognition. The development of preprocessing techniques has generally stemmed from a desire to make characters fit a human abstraction of how they should look. The assumption is that if characters all conform to a human ideal then recognition will be easier and more accurate. However there is no clear reason to believe this assumption is true. Neither is it clear what the human ideals should be. Many of them appear to be little more than aesthetic forms with no real link to the distinctive features that make them recognizable by the classifier. The preprocessing approach has not yet led to the near perfect accuracy that might be expected if an idealized character representation was obtained. It is therefore possible that either the assumption is incorrect or that preprocessing techniques may be unable to produce this idealized form.

It is my belief that both of these statements are true and therefore abstracting a human researcher's intuitive knowledge into a recognition system may not be the best approach. The variation in human handwriting styles is immense; compacting this into a simple, general set of rules is extremely difficult. A computer, basing its simplifications on a large set of exemplar data, may be a much more reliable judge than a human, basing abstractions on preconceived ideals. The computer will produce rules based on what characters *actually* look like, whereas humans tend to base rules on what they think the characters *should* look like. These human assumptions are prone to misconception and oversimplification. In addition, some of the human rules that have appeared are based on abstractions of how humans form characters rather than on how they recognize them. The two are not necessarily the same.

Another reason for questioning the application of these techniques to character recognition is that many of them were originally developed for use on other types of images, for example, engineering drawings and maps. While they have proved useful in some of these applications, where the emphasis is often on reduction of data rather than on recognition, it does not necessarily follow that they are suitable for character recognition, where preservation of distinguishing detail is the primary concern. There is always a risk, when altering the initial image data, that information will be lost or errors will be introduced. Once preprocessing has been performed, it is generally not possible to retrieve lost information or correct errors. There is therefore good cause to question the use of preprocessing techniques which lose information.

Another reason to question their use is that the effects of some preprocessing techniques are achieved in other ways at different stages of character recognition systems, for example, during feature extraction or classification. We must therefore question the need for an additional stage which may simply be duplicating the effort.

In evaluating preprocessing techniques, the primary factor to be considered is the effect on classifier recognition rates using the subsequently extracted features. The recognition rate is influenced by several possible effects of the preprocessing: the clustering of similar characters, the introduction of errors and the loss of information. Another important factor is their suitability for use in a real application, in particular their extension to larger character sets and noisier data. Other factors considered here include the reduction of data needed to store the characters, the computational cost involved in the process and the possibilities for achieving the same effect using alternative methods.

While many papers have been published on preprocessing techniques, few have questioned their application to the character recognition problem or compared them to alternatives. One paper which has is Güdesen's [Güd76] which shall be referred to several times in this chapter.

Güdesen compared a range of preprocessing techniques on simply segmented 9x14 pixel bitmaps of handprinted characters. He extracted the first 30 coefficients of the Karhunen-Loeve expansion (see section 2.5.2) as a feature vector. Classification was by a parametric Bayesian quadratic discriminant assuming a multivariate Gaussian distribution (see section 6.2.1). The test data was obtained from 200 writers who were trained to print 15 rows of model characters. A set of 560 examples of each digit (0-9) was divided in two — half for training the classifier and half for testing it. Güdesen claimed that a larger set of digits and upper case characters, with 3000 examples of each, gave only slight differences from the results on the smaller set. With no preprocessing the digit set gave an error rate of 1.7% with no rejection.

In considering Güdesen's results and conclusions we must take note of several features of his experiments. Firstly the error rate with no preprocessing is very low. This is due to the high quality of Güdesen's data. The use of special forms for writing on and a clean environment allows noise to be kept to a minimum. The use of model characters to copy considerably reduces the style, size, position and slant variability of the data. Given that the intention of Güdesen's experiment was to compare techniques which are intended to reduce this variability, the use of such constrained data was perhaps not ideal. However, the choice of data reflected the capabilities of handprinted character recognition at the time. It should also be noted that only digits were evaluated in the full test and the preprocessing methods may perform differently on more difficult character sets such as lower case letters.

The type of feature is also a significant factor in Güdesen's experiment. Karhunen-Loeve expansions are used in statistical classification whereas some of the preprocessing techniques (thinning and polygonal approximation) are generally intended for structural recognition. It is difficult, however, to quantitatively compare techniques using different classifiers. Güdesen based his evaluation of techniques almost entirely on the statistical classification error; his results were therefore biased against some techniques.

4.3. Pros and Cons of Preprocessing

One of the general advantages of preprocessing is that in many cases it reduces the amount of data needed to represent characters. This can, in some cases, reduce the computational cost of the subsequent feature extraction or classification processes. However, this advantage must be weighed against the additional cost of the processing itself. Preprocessing operations involve at least one pass through the original character data. Some operations (the ones that produce the greatest reduction in data) can require multiple passes. If the feature extraction process only requires a single pass, which several of them do, then there is no cost benefit in simplifying the representation.

If preprocessing succeeded in its aim of making characters fit idealized abstract forms then it would essentially perform a perfect classification. The problem, however, is that most techniques encounter problems where preprocessing should produce one result in the context of one character and a different result in another character. The problem cannot be accurately resolved without knowing what the character is. In other words, although perfect preprocessing would produce perfect classification, it isn't possible to perform perfect preprocessing without already knowing the classification.

While most researchers would agree that perfect idealization of characters is not an achievable goal, preprocessing should, to an extent, aid the clustering of character classes and produce better classification results. In practice, this does not always happen. There are two reasons why preprocessing can fail to produce better results. Firstly, irreversible errors and distortion can be introduced into the characters which cause them to be mis-classified.

Secondly, preprocessing strategies tend to be designed with the aim of making characters of the same class more similar. What they should really be aiming for is the separation of classes. The simplification resulting from preprocessing tends to make all characters more alike. Thus the fact that characters of the same class are more alike does not necessarily mean they are easier to distinguish from other classes. In some cases the opposite can happen: simplification causes more class overlap than before, making classification more difficult.

Now let us consider the benefits and weaknesses of specific preprocessing techniques.

4.3.1. Thinning and Vectorization

Thinning and vectorization (see section 2.3.1) are perhaps the most obvious attempts at making characters fit human abstract forms. Based on the assumption that characters are best recognized from a widthless representation, thinning and vectorization aim to reduce a character to either a unit-width pixel skeleton or a zero-width line (vector) skeleton.

This assumption is difficult to justify. It has not been proven that biological thinning is used by the human brain. While it can be shown that certain neurons in the human visual cortex detect lines at specific orientations, in the first instance these will be edges (contours) of objects, detected along contrast borders in the eye's receptive field. It does not necessarily follow that these edge-based image representations are subsequently thinned. Indeed it is possible that human recognition is based on the actual shape outlines that are seen, rather than abstract thinned forms. Outlines would seem a more robust basis for visual recognition since they can represent images other than characters, and shapes that do not have a useful thinned equivalent, such as circles and ellipses. It is probable that characters are processed differently to other images, after being initially identified as text, but there is still no evidence that thinned representations are used.

While it can be argued that humans write using strokes rather than outlines, it does not necessarily follow that the way we write and the way we recognize characters are the same. In fact, the human brain uses different cortical structures for these two tasks, as evidenced by people with certain neurological disorders. Some patients are able to read but not write (pure agraphia) (e.g., [RAB89]), while others can write but not read (pure alexia) (e.g.,

[QG92]). The use of different parts of the brain suggests that different processes may be involved for recognition of characters and drawing of characters.

In the absence of biological plausibility we must look to machine recognition for a justification of the assumption. However, examination of the thinning/vectorizing approach suggests that these processes have a detrimental effect on recognition accuracy. As discussed in section 2.3.1, the main problems with thinning and vectorization techniques are their loss of geometric information, their sensitivity to noise and the difficulty of correct skeletonization in areas of overlapping pen strokes.

Although skeletal representations have been used for conventional, feature-based, statistical character classification they do not give high accuracy results when used in this way. This is because they do not accurately preserve the geometry of characters, preferring to rely mainly on structure [LLS92]. The medial lines formed by the process are only approximations of the pen strokes. They essentially average the outlines on either side so they are only really accurate when the outlines are precisely parallel. It is common to smooth the vectors, which removes more of the detail.

For statistical classification, using the types of features described in sections 2.5.1 - 2.5.3, the geometric detail of the shape is essential for accurate recognition. Section 5.6 compares the performance of such a feature extracted from outlines and their vectorized forms, and shows that the vectors preserve less of the discriminatory detail. The Karhunen-Loeve features used by Güdesen (see section 4.2) also had problems with thinned characters. This was partly due to the loss of geometric information but also due to another major problem with skeletons, previously mentioned in section 2.3.5. While the aim of early thinning methods was to make similar characters more alike by unifying their line thicknesses, it actually made *all* characters more alike. Güdesen observed that from a statistical viewpoint thinning brings the class means closer together while at the same time making the class variances greater. Statistical discrimination of classes using these types of feature becomes much more difficult as a result of thinning and vectorization.

The loss of geometric information is not regarded as a problem by most researchers as the skeletal representations are intended to be used for structural classification rather than statistical. Skeletons facilitate the extraction of higher level features or primitives, such as lines and curves or other types of stroke, and their structural relationships. The effectiveness of thinning and vectorization should not therefore be judged by their unsuitability for statistical classification. Rather it depends on the accuracy of their primitive extraction.

Noise frequently produces spurs on the skeleton which should not be present. This creates problems for the feature extractor and classifier, more so than noise in outline-based image representations. Large spurs can be caused by very small outcroppings of noise, particularly with distance transform methods. Components of an image which should be separate may be joined by noise. The preservation of connectivity, which is the focus of most thinning techniques, will mean these components are treated as a single component in the skeletal representation; strokes may therefore become merged.

In addition to affecting the geometry of characters, these spurs and joins introduce significant errors into higher level feature approaches. Geometric and topological features (see section 2.5.4) are already very error-prone; introducing new difficulties through thinning and vectorization means the reliability of detecting lines, curves, corners, junctions, etc. is undesirably low. Similarly the extraction of primitives for higher level structural classification (see section 6.3) is subject to a large degree of error. Unwanted spurs will probably be extracted as individual primitives. Unwanted joins can merge primitives together. Although most structural classifiers have some degree of tolerance to primitive extraction errors, they are still highly undesirable. By allowing for error tolerance the classifiers also allow greater structural flexibility in correctly represented characters. This degree of flexibility creates greater amounts of class overlap within the classifier and therefore increases the classification error due to ambiguity. The simplified primitives used in structural character classification make classes very similar even without the error tolerance; with it, the degree of overlap is often so great as to make disambiguation by contextual validation extremely difficult.

It seems unlikely that the errors in extraction via pixel erosion or distance transform techniques can be reduced. Vectorization techniques offer the possibility of more accurate primitive extraction but some errors must be expected. The difficulty of accurate structural representation of characters remains a fundamental problem with thinning and vectorization, and the structural recognition approach. It should be noted that while structural character recognition has been investigated for many years, it has produced few results comparable to those achieved by statistical classifiers, or, in recent years, by neural classifiers. The computational cost of structural methods (which are template matching procedures) is also a major handicap of the approach. Feature-based statistical or neural classification are much better suited to practical character recognition applications. Chapter 6 compares these classification approaches in more detail.

A further problem with the thinning/vectorization approach is the inability to accurately thin or vectorize areas where pen strokes overlap. Areas where strokes double back on themselves or cross other strokes are extremely difficult to vectorize as they are often indistinguishable from single pen strokes. The correct path of strokes across junctions is often ambiguous and depends on the class of the character being vectorized. It is therefore almost impossible to correctly thin or vectorize these areas without knowing in advance what the character is. Although some progress has been made in improving the vectorization of junctions (see section 3.8) no existing vectorization algorithm has attempted to vectorize large areas where strokes double back over themselves (except where they form junctions, in which case the vectorizations are nearly always wrong). The only practical strategy is to treat them as single strokes. Stroke-based representation of these characters is therefore inherently inaccurate.

As mentioned in the previous chapter, Pavlidis states that "strict vectorization is not sufficient for a practical character recognition system" [Pav86]. The loss of shape detail makes the approach unsuitable for feature-based recognition. The inaccuracies in representation and the difficulties they introduce into structural classification make accurate structure-based recognition improbable also. If these techniques are to be successful then significant improvements must be made to enable preservation of detail and severe restriction of the number of extraction errors. A hybrid classification approach would appear to be necessary.

4.3.2. Smoothing, Filling and Joining

Smoothing (see section 2.3.4), and the related techniques of filling and joining (section 2.3.3), are commonly applied to all forms of image representation. The techniques are in some cases different but achieve similar results. Their aim is to remove noise and distortions from character images, thereby making characters of the same class more similar. Approaches to smoothing often stem from a desire to make the characters more aesthetically pleasing to humans by removing bumps or jagged edges, and simplifying the representation.

Reduction of the data required to store characters is a desirable, though not essential, quality of smoothing processes, provided important detail is not lost. The removal of noise from a representation is also desirable as it is likely to reduce errors in feature extraction and classification, provided, again, that important detail is not removed along with the noise. In practice, however, smoothing cannot distinguish between noise and detail, so these provisions are not met.

The levels of noise in real character recognition applications are sufficiently high that they prohibit a size-based distinction between noise and detail. Although the most powerful methods can be tailored to form a precise size-based division between bumps to be removed and bumps to be kept (e.g., Ho and Dyer's medial axis-based smoothing [HD86]) there is no perfect cut-off point. Smoothing methods based on the size of protrusions or intrusions in a character have to make a compromise between the amount of noise they aim to remove and the amount of incorrect removal of detail they will allow. Any loss of detail at this stage is detrimental to the classification rate. Even small protrusions from a shape can indicate the difference between two character classes, e.g., a small tail at the bottom left of a circle might indicate the character is an 'ɑ' rather than an 'o'. On other characters, a similar sized protrusion might easily be considered to be noise.

Shape-based smoothing methods (i.e., methods which look for particular patterns of protrusions and intrusions) are also incapable of distinguishing detail from noise for the same reason. There is no way to tell whether a protrusion is noise or detail without knowing in advance what the character is. Filling and joining techniques face the same problem. A small black dot is usually to be removed as noise by the filling process, but if it is the dot of an 'i' or 'j' then it is a vital piece of distinguishing detail. Joining techniques which merge all of a character's components into a single component are also fundamentally unsound for processing lower case 'i's and 'j's. Merging the dot of the 'i' with its stalk renders it indistinguishable from an 'l'. But in other circumstances, the joining of components is appropriate and beneficial. Banks [Ban91], whose joining method was described in section 2.3.3, concludes that there is no perfect definition of when two components should be joined because identical components should be merged in one context and not in another.

All these techniques perform acceptably when the character image meets their expectations. However, no algorithm can cater for every situation without prior knowledge of the character's class. Since this knowledge is unavailable at the preprocessing stage, the decisions about what to treat as noise and what to treat as detail, and when to join components and when to leave them separate, should be left to a later stage when we have some idea of what the classification is.

4.3.3. Polygonal Approximation

Polygonal approximation (see section 2.3.2), when considered as a preprocessing stage rather than a feature extraction, aims to produce a smoothed outline form. It suffers from the same problems as other types of smoothing. There is no clear division between noise and detail. Polygonal approximation loses more information than most other smoothing methods because it does not just remove noise; it simplifies the character's shape to a small set of lines. It therefore faces the additional problem of deciding which portions of the outline should be preserved in detail and which should be simplified. As with the noise problem, there is no general way to decide this without prior knowledge of the character's class.

The polygonal approximation technique is more commonly used in applications where the objects are larger, such as the representation of maps or engineering drawings. In these applications, the technique is appropriate because the fine detail is less important; simplification and removal of noise is the main concern. Polygonal approximation performs well at these tasks in these applications because the objects often have long sections of roughly straight or smoothly curving lines. The smaller objects that are typical in character recognition do not benefit greatly from this kind of smoothing. Their outlines generally vary much more over short distances and therefore require a large number of approximating line sections to accurately represent them.

In character recognition, polygonal approximation is not commonly used for smoothing alone. Other smoothing techniques, that concentrate only on removing small, local pieces of noise, are generally preferred as they risk less distortion of the characters. Approximations are best at smoothing when the images are clean and smooth initially. In these cases, the only benefit is a simpler representation, but this rarely justifies the computational cost of the procedure.

Polygonal approximations are more commonly used as structural feature detectors or primitive extractors (see section 6.3.2). Although their use as features has been discussed in other sections, we shall consider these primitives here from a preprocessing viewpoint. Polygonal approximation primitives are less prone to errors than stroke-based primitives taken from thinned or vectorized characters. However, they have had no success as pure structural primitives and so have mainly been used in hybrid statistical/structural methods, such as attributed string matching, where the primitives have associated features (e.g., position, angle, length). Attributed string matching with polygonal approximations has had some success on simple shapes and handwritten digits but is unlikely to be useful for more general character recognition. This is because there is little similarity between characters of the same class.

The vertices of the polygons, and hence the lengths and angles of line sections, are highly variable between similar shapes. During the split and merge procedure (see section 2.3.2), vertices often become stuck in small local distortions (bumps and notches) on the outlines where they would not normally occur. Though they are actually a way of smoothing noise, they are themselves affected by it. Even without these local distortions, the number of ways of approximating curved line sections is very large and the approximation algorithms cannot guarantee that the same vertex points will be found for every character. They cannot even guarantee that the same *number* of vertex points will be used for a particular curve of a character.

The total number of approximating lines used has a significant effect on the variability of approximations. There are two options for choosing how many lines to use (equivalently how many vertices to use):

1) take the same number for all characters;

2) take a variable number according to the needs of the approximation algorithm.

Using the same number for each character presents difficulties. The number of lines must be sufficient to represent characters with complex outlines, such as 's' or 'm'. Approximating much simpler characters, such as 'l' or '1', with this many lines gives a much greater accuracy of representation than intended and allows much greater variability between simple characters. To overcome this it is important to consider the length of line primitives when evaluating the cost of edit operations during matching. This is why pure structural methods have failed and attributed primitives are required.

Using variable numbers of lines means that the number of primitives will vary between characters of the same class. Insertion and deletion of primitives must therefore be tolerated at reasonably low costs. This causes the same problems as error tolerance in stroke-based primitives. In practice it is extremely difficult to find costs for edit operations that allow flexibility within a class but do not allow overlap with other classes. This is why the approach has only been successful with simple shapes and small sets of classes. The degree of variability and the tolerance requirements cause too much class overlap when used on wider recognition problems such as upper and lower case letters.

Another fundamental problem with the use of polygonal approximations as structural primitives is that they represent outline structure by their ordering. This works for single outlines but there is no obvious way to represent characters consisting of multiple outlines. There is no simple ordering that will allow multiple outline approximations to be compared with their corresponding approximations in a template. Broken characters in particular are very difficult to match to an unbroken character template of the same class.

Approximation-based primitive extraction also faces the same problem of the structural approach that was described in section 4.3.1 (and also in chapter 6), namely, the high computational cost of structural classification methods.

4.3.4. Normalization

Normalization techniques (see section 2.3.5) can loosely be divided into two types: deforming (line-width normalization, and slant and slope correction by shearing) and non-deforming (size and position normalization, and slant and slope correction by rotation). By deforming we mean that the shape of a character is changed by the normalization process.

In the same way that smoothing, filling and joining present problems to the recognition task, deforming normalization also has undesirable effects. Non-deforming normalization, on the other hand, is generally beneficial.

Scaling the character whilst precisely preserving its shape does not normally have any negative effect on the discriminatory detail of the image representation. Many of the feature systems used in character recognition are intentionally size independent. It is one of the fundamental assumptions of the field that characters should be able to be recognized regardless of their size.

Long-term problems do arise however if all characters are pre-scaled to a uniform size. When the class alphabet includes both upper and lower case characters we find that the upper and lower case forms of some letters are very similar, even identical, in shape and differ only in their height and width, e.g., ‘C’ and ‘c’, ‘O’ and ‘o’, ‘S’ and ‘s’, and ‘V’ and ‘v’. A common feature that is often used in conjunction with size-independent features measures the original height and width of the character. This information is useful in distinguishing these similar upper and lower case characters. The problem with size normalization therefore, is that the original height and width information is lost if the character is scaled prior to feature extraction. Preserving the scale factors used in size normalizing each character, and storing these to be passed on to the feature extractor, is one way of circumventing this problem.

Another problem with size normalization is that in some representation formats the scaling may not precisely preserve the original shape. Quantization errors occur when bitmaps are scaled on a discrete grid unless the scale factor is an integer. The actual scale factors required for practical size normalization are almost always non-integers so some distortion is inevitable. In this case the size normalization is not truly non-deforming. Güdesen (see section 4.2) was as precise as possible in his size normalization. His method, described in section 2.3.5, calculates the amount by which scaled pixels overlap squares on the bitmap grid, and thresholds this amount to determine whether the squares should be black or white in the new representation. This still introduces quantization error however. Despite this, Güdesen reduced his original error rate by a factor of six using this normalization process, clearly showing that the improved classification accuracy resulting from the standardization of character size far outweighs the errors introduced by imprecise scaling.

Position normalization (centering or displacing) is another (generally) non-deforming normalization process, whose sole purpose is to aid the feature extraction process. By aligning characters in analogous positions within a reference frame the process overcomes the problems of some feature systems. Several of the feature systems in early usage were not invariant to the position of characters within the reference frames in which they were presented (usually bitmaps). This is less of a problem with modern systems, most of which use more flexible formats that are not related to individual frames for each character but are related instead to the frame of the entire document, e.g., outlines and vectors. Rather than having them predetermined by the form of presentation, these feature systems generally determine their own reference points, e.g., centre of gravity, or uppermost left-hand point on an outline.

For feature systems which are still limited by individual character reference frames some form of position normalization is desirable to provide a degree of position invariance. Displacement is a suitable technique for this. Centering is suitable provided it is a non-deforming process. The centering transformation should translate by integer amounts if a discrete representation is being used. Non-integer translations on a discrete grid introduce the same kind of quantization errors as size normalization, and these should be avoided. Güdesen noted that his centering techniques sometimes translated parts of the character outside the reference frame. This should also be avoided as it is these outermost parts of characters that are likely to hold the important shape details necessary for identification.

Skew correction by rotation is a non-deforming operation, provided the character is translated to continuous coordinates. Rotation to discrete (integer) coordinates introduces a large number of quantization errors. Güdesen used a discrete coordinate grid and found that these errors could "tear a character apart and give rise to substantial structural changes." The overall classification error increased by a factor of 4 in Güdesen's experiment. He suggests that the use of larger grids would reduce the error but the real problem is the size of the characters. To reduce quantization error, the characters must appear larger on the grid. This requires higher resolution scans. The use of continuous (floating point) coordinates is a far better solution.

This form of correction can be achieved in an alternative way, however, by using rotation invariant features, e.g., global transforms (see section 2.5.2), geometric and topological features (see section 2.5.4).

Slant correction by shearing is deforming but is generally the most effective of these types of operation. Few feature or classification methods can achieve exactly the same effect (they can correct slant by rotation but not by shearing). Although it is not possible to estimate the angle of slant with 100% accuracy, shearing can still produce approximately upright characters. Results on handwritten digit recognition using the Radial Distance/Sector Area features were improved by the addition of shearing slant correction (see section 5.5.6). This suggests that the benefits of the sheared forms outweighed the distortion effect of the process. Güdesen was less impressed with shearing. He regarded it as computationally expensive and, although he reduced his error rate by shearing in the x direction, he thought the calculation effort barely justified the improvement. Güdesen also found that additionally shearing in the y direction was ineffective. The errors introduced by the distortion cancelled out the advantages of slant removal and the error rate stayed the same as it was without any slant correction. However, the distortion in Güdesen's experiments was partly due to quantization errors caused by his use of discrete coordinates. With continuous coordinates the distortion would not be as great, although shape can still be deformed significantly in heavily slanted characters.

There are long term problems with slant correction because some characters are naturally slanted, e.g., '\', '/', '7'. These characters will become harder to distinguish if this natural slant is removed, particularly the forward and backward slashes which become indistinguishable from some 'l's and '1's. This problem could be dealt with by storing the amount of the original slant as a feature. Another way is to use classes which represent all these possible characters ('\', '/', 'l' and '1') and disambiguate them later according to the context in which they are found. This is not a great problem as such a strategy will probably be required anyway to disambiguate '1's and 'l's. However, it is hard to see how forward and backward slashes can be distinguished by context. A third approach is not to shear the characters but instead to learn the range of slant for each character through the use of a large, representative training set.

Shearing of whole words is perhaps the most effective strategy. If a reasonably accurate measure of slant can be determined for the whole word, then individual characters with no natural slant may be sheared upright, while the other characters in the word retain their natural slant. Shearing of words can also aid their segmentation by vertical lines. The difficulty is in determining the correct amount of shearing required. Bozinovic and Srihari's method [BS89] of shearing cursive script is reasonably effective but performs best when characters have consistent slant within each word. Words with variable character slant cannot be appropriately sheared by a global operation. The variably slanted characters must be sheared individually after segmentation.

Line-width normalization is rarely used as it can produce too much distortion and loss of detail in characters. While the specific methods described in section 2.3.5 have had some practical use in their particular systems, they are not suitable for general application.

4.4. The Alternative — Minimal Preprocessing

The underlying assumption of the traditional preprocessing approach is that there is something "wrong" with input characters that do not conform to a small set of idealized class models. A recognizable character should exhibit no distortion, noise, slant, style variation or variability of line thickness. This section suggests that an alternative philosophy, which rejects that assumption, may be equally effective, if not more so. The minimal preprocessing approach assumes that any input character which can be recognized by a human is a valid representation of its class. The approach attempts to avoid the time consuming and potentially distorting preprocessing stages and leave any normalization or smoothing to the later stages of recognition.

Early character recognition used simple features and classifiers. Some preprocessing of the input was required to achieve worthwhile results. However, as feature extraction and classification techniques developed they became more tolerant to noise, distortion and variation in characters. This resulted in a standard model of character recognition where this tolerance was being attempted at all stages of the process. The fundamental idea behind the minimal preprocessing approach is that this kind of tolerance is simpler, more effective and less error-prone when performed at the later stages of recognition. A parallel can be seen with Casey and Nagy's approach to segmentation [CN82], where a blind, single-pass attempt is rejected in favour of a recursive system, linked to the later stages of classification and contextual processing.

We shall now see how preprocessing can be avoided by using alternative approaches. There are four main functions of preprocessing: the facilitation of structural primitive extraction; simplification of the character representation; removal of noise and distortion; and normalization.

Although the use of thinning, vectorization and polygonal approximation to facilitate primitive extraction can introduce errors, it is unlikely that alternative, outline or bitmap-based, extraction techniques would be less error-prone. The difficulty of tolerating extraction errors, without allowing large amounts of class overlap, is a fundamental problem in structural classification. Rather than attempt to address this problem, the minimal preprocessing approach favours a statistical or neural classification, using the types of feature described in section 2.5. These classification paradigms are much faster and therefore far more practical than structural recognition. Error tolerance causes much less class overlap in feature-based classification and these approaches are generally more accurate than structural methods.

Thinning, vectorization and polygonal approximation can still be used to simplify the image representation for presentation to statistical and neural classifiers. However, this information-losing and error-prone simplification is not well suited to feature extraction, where preservation of detail is essential. These preprocessing operations, and their associated errors, can be avoided by using the original outline or bitmap image representations as they are. Although they are not simplified like skeletons and approximations, this is not a major factor in modern character recognition. Modern computers do not have the same restrictions on memory and speed that made simplification desirable in early OCR Research.

Many features have been suggested for use on outlines and bitmaps (see section 2.5). Several of these features have certain invariant properties that can obviate the need for other types of preprocessing (smoothing and normalization). This makes them ideally suited to the minimal preprocessing approach.

Smoothing (including the smoothing of outlines by polygonal approximation) is intended to remove noise and reduce distortion of the characters. The alternative approach is to tolerate noise and distortion in the later stages. Many types of feature have been developed with noise tolerance in mind. Character classifiers generally exhibit some degree of noise and distortion tolerance; many of them are capable of performing correct classification in reasonably noisy environments. The combination of noise tolerant features with noise and distortion tolerant classifiers provides an effective alternative to smoothing.

The difficulty of removing noise without removing important detail suggests that it is better to leave the image as it is and allow the later recognition stages to decide which is which. Smoothing processes remove protrusions and intrusions uniformly over the image, with no regard to their positions on the character. These positions can often indicate whether the objects are detail or noise. A significant advantage of tolerating noise at the feature and classifier level is that the classifier's attention focuses on the areas of the image which typically contain the distinguishing detail for specific classes. They therefore emphasize detail if it is in the right place and de-emphasize noise in the wrong place. While it is possible that a noisy protrusion in a character occurs in the position where a distinguishing protrusion is expected in a character of a different class, this is less common. Most of the time the minimal preprocessing approach will be more effective than smoothing.

Filling and joining are considered detrimental to recognition as they introduce too many errors. The minimal preprocessing approach favours the use of noise tolerant features and classifiers to achieve the same effect as filling. Noise tolerance at these later stages is less error-prone and more efficient for the same reasons as for smoothing. Joining requirements can be avoided by using features which can operate on broken and multi-part characters. This is an important capability for most systems because accurate joining is impossible without knowing what the character is. Many features do not have this capability so devising such a feature is an important aim of the minimal preprocessing approach.

Features have been developed which can provide alternative methods of normalizing characters. The main advantages of using invariant or normalizing features are that they are usually faster and do not require storage of the normalized image representation. Other advantages may also arise for specific types of normalization.

Size normalization can be achieved at the feature extraction stage. Since many of the more successful feature systems are naturally size independent, the normalization process of scaling and storing the image representation is unnecessary. In addition, the advantage of leaving size invariance to the feature extractor is that the original height and width information is still available at the extraction stage, without requiring it to be specifically preserved. Thus no preprocessing size normalization is required if a suitable feature system is used.

Further advantages may occur where the original image representation uses integer coordinates, e.g., an unpreprocessed outline. A precise preservation of the character's shape will very often require scaling these integers to floating point coordinates. This increases the computation time of the feature extraction. The alternative strategy might allow the feature extractor to operate on integers and hence be faster.

Position normalization can also be achieved at the feature extraction stage. By using a more flexible representation format than bitmaps the need for alignment of each character within an individual reference frame is eliminated. Many feature systems can determine their own reference point for producing position invariant features. The translation and storage of a new representation of each character is therefore a redundant process. Again, the choice of a suitable position invariant feature system obviates the need for this kind of preprocessing.

Slant correction by shearing is not easy to perform during feature extraction, however the similar rotational correction can be achieved using rotation invariant features. Since features are usually continuous-valued, they can cater for rotation without introducing quantization errors, such as can occur with preprocessing rotation in discrete coordinates. As with the previous forms of normalization, the use of rotation invariant features is more efficient than preprocessing because the correction is implicit, rather than requiring an explicit translation and storage of each character.

While some forms of character variability can be tolerated by the use of appropriate features, other types of variation, such as style and slant (shear), must be dealt with by the later classification stages. The minimal preprocessing approach therefore requires that a large training set is used which represents the range of variability. Although the required set is large, databases such as the NIST set (see section 2.6.1) exist which give an acceptable representation of the natural variation in real handwriting. Not every conceivable variation may be included, but a system trained for general recognition from such a set will learn the most common ranges of variation. It is likely that practical character recognizers will have to be able to learn new users' handwriting, either while they are in use or from new training samples taken from individual writers. Assuming that a system can be tailored to learn an individual user's writing, it is not necessary for the initial training set to include every possible variation.

The minimal preprocessing approach has been adopted in the remaining work in this thesis. Slant correction is the only preprocessing operation which has not been entirely avoided. Unprocessed outlines are used rather than thinned or vectorized representations. Chapter 5 contributes a new feature to this approach which fulfills the requirements mentioned above. The large NIST and CEDAR databases are used to teach classifiers the wide ranges of character variability without simplification or abstraction by preprocessing. The results obtained using this approach suggest it is a viable alternative to traditional preprocessing.

4.5. Conclusions

The traditional approach to character recognition, involving one or more one-way preprocessing stages, has been discussed. Problems with the approach have been described and an alternative, minimal preprocessing approach has been proposed which aims to avoid these problems.

A statistical or neural, feature-based classification is favoured over structural methods. An outline or bitmap image representation is used, rather than performing a thinning or vectorizing preprocessing operation. Features must be selected which are tolerant to noise and variations in orientation, size and position. Some tolerance to distortion and style variation, and an ability to operate on broken or multi-part characters, is also desirable. This avoids the need for the preprocessing stages of smoothing, filling, joining and normalization. Classifiers should also be tolerant to noise and distortion. The use of large, representative training sets is required, to cope with the wide range of character styles.

There are too many different preprocessing methods to allow a quantitative comparison of all of them and no such evaluation has been attempted. While no conclusion can be reached about which approach is better, it is suggested that minimal preprocessing avoids many problems, is computationally less expensive and has the potential to be at least as effective as the preprocessing strategy.

Chapter 5

A Robust Feature for Outline Representation

5.1. Summary

Following the discussion in chapter 4, this chapter proposes a new outline-based feature for off-line character recognition, designed with the minimal preprocessing approach in mind. Reasonable invariance and noise tolerance are attained without preprocessing, though the feature can also be extracted from preprocessed outlines. The main quality of the feature is its ability to cope with broken outlines and multi-outline characters, which many outline-based features cannot.

Section 5.2 outlines the problem and the work on which the new feature is based. Section 5.3 describes the feature and section 5.4 expands on its extraction from outlines. Some results on the NIST and CEDAR segmented character databases are reported in sections 5.5 and 5.6, the latter comparing classification accuracy on preprocessed and non-preprocessed characters. The possibility of adaptive segmentation through the use of these features is explored in section 5.7. Extensions to the feature extraction and adaptive segmentation methods are proposed in section 5.8. The success, usefulness and potential of the feature is evaluated in section 5.9, and section 5.10 summarizes what has been achieved. Section 5.11 describes the notation and symbols used in this chapter.

5.2. Normalized Contour-Based Feature Strings

As described in section 2.5.2, numerous features have been proposed for the representation of outlines to a character recognition system. In section 2.5 several abilities of ideal features were listed. In practice, not all of these are achieved by any one feature. In 1990 Dinstein and Landau [DL90a] proposed a feature for object recognition, the normalized contour-based feature string, which had many of the desired properties.

The normalized contour-based feature string is a list of distances from the centre of gravity of an outline to a point on the outline, taken at fixed intervals along the contour. The fixed distance is either a function of the maximum centre-of-gravity-to-outline distance for the contour, or it is a factor of the total length of the outline. The second method is preferred as it gives a constant number of feature points for each character. With the first method the number of intervals varies, with longer perimeters producing longer feature strings; this complicates matching. The distances are normalized using the maximum distance as the normalization factor.

This feature is invariant to position, size and rotation (assuming an analogous starting point can be determined for each character). Dinstein *et al.* [DLG91] describe how it can be computed in parallel and how an approximation of the original outline can be reconstructed from the feature strings. The main quality this feature lacks is the ability to handle multi-outline characters (e.g., 'i') and broken images which can be caused by binarization thresholding errors, pens with poor ink flow, hasty writing and other factors.

A variation on the Dinstein and Landau approach is proposed which maintains the properties of the feature and in addition is tolerant to breaks in outlines and can easily operate on multi-outline characters. It is therefore more robust and provides a solid basis for character recognition applications. It has also been extended to reduce the loss of detail between the sampled intervals, thereby increasing the accuracy of the representation.

5.3. The Radial Distance/Sector Area Feature

The Radial Distance/Sector Area (RD/SA) feature is a composite of two closely related features. The Radial Distance feature and Sector Area feature are (normally) equal length, scaled, floating point vectors. The first is a simplified geometric representation of the outer edge of an outline and the latter is a measure of the distribution of the character's area. Although the two measures could be used separately, they are intended to be combined as a single feature vector.

The Radial Distance feature takes measurements from a set of nested outline loops (see section 2.1.3), representing the pixel boundaries in a text segment. Positional invariance is achieved by basing measurements on a central reference point for each segment. Feature measurements will then be independent of where the character segment occurs in the image. Two methods have been tested for determining the centre point.

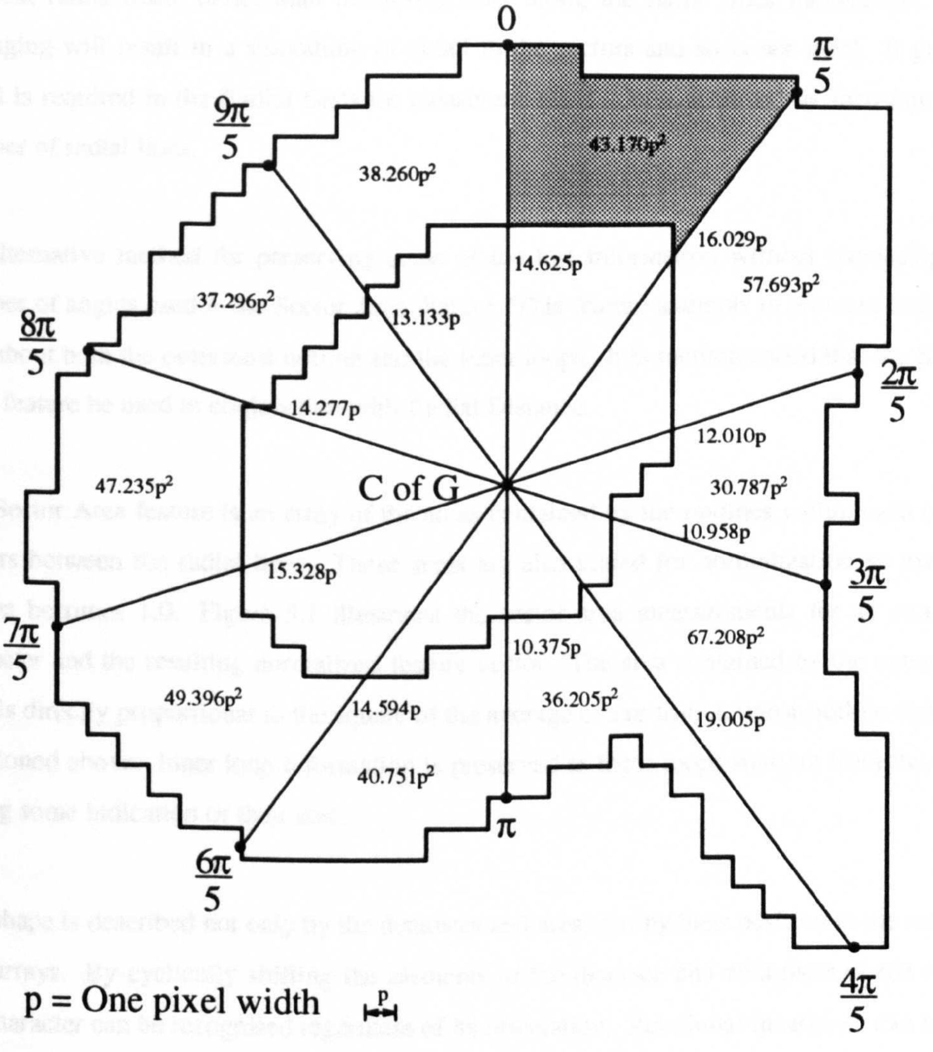
The greatest invariance of centre point was achieved using the *centre of gravity* method. The centre point is simply the centre of gravity of the segment. All even level outline loops in the segment are considered, remembering that odd level child loops subtract from the area of the even level parent loop. Uniform weight per unit area of the outline is assumed, except in situations where the segment has been vectorized and area information has been lost, in which case uniform weight per unit length of the vector is assumed.

Note that other forms of preprocessing, such as smoothing and line width normalization, can result in outline loops with zero area. In the context of other loops with non-zero area this causes a problem. Such loops must be taken to have zero weight which means they are ignored for the purposes of determining the reference point. By avoiding these forms of preprocessing, it can be ensured that all outline loops have a non-zero area.

The second method, called the *centre of segment* method, is to find the maximum range of x and y coordinates covered by the outlines in the segment. The midpoints of these two ranges give the coordinates of the centre point. This method is inferior to the centre of gravity method when the original outline area information is available, but proved far more effective on vectorized characters. The centre of segment varies more between similar characters than the centre of gravity but is less likely to fall on a vector. This is beneficial to the method, as will be explained later.

The Radial Distance feature is an array whose elements are the furthest Euclidean distance from the centre point to a point on an outline loop, along a radial line extending from the centre point at a particular angle from the vertical. The array elements are ordered by the size of the angles, which encompass the range 0 to 2π in equal steps. The Euclidean distances are scaled for normalization so that the largest distance becomes 1.0. (Scaling so that the average distance becomes 1.0 was less effective.) This gives a size invariant feature vector. The representation of the outline increases in accuracy with the number of angles.

Although the detail of the inner loops is lost, most of a character's shape information is present in the outermost outlines. However, information about the outlines in between the radial lines is also lost. It would be possible to include this information by averaging the centre-to-outermost-outline distances over the whole range of angles in the sectors between



Radial distances	14.625	16.029	12.010	10.958	19.005	10.375	14.594	15.328	14.277	13.133
Sector areas	43.170	57.693	30.787	67.208	36.205	40.751	49.396	47.235	37.296	38.260
Normalized RD	0.770	0.843	0.632	0.577	1.000	0.546	0.768	0.807	0.751	0.691
Normalized SA	0.642	0.858	0.458	1.000	0.539	0.606	0.735	0.703	0.555	0.569

Figure 5.1 Example of Radial Distance/Sector Area feature measurements. Ten angles in steps of $\pi/5$ have been used and the centre of gravity has been used as the reference point. The shaded region indicates the area of the image contained within the first sector. The measurements are distances in pixel widths (p) and areas in square pixel widths (p^2). The corresponding normalized feature values are shown in the table.

adjacent radial lines, rather than measuring only along the radial lines themselves. This averaging will result in a smoothing of detail in the sectors and so is not ideal. If greater detail is required in the Radial Distance measurements it is best achieved by increasing the number of radial lines.

An alternative method for preserving some of the lost information without increasing the number of angles used is the Sector Area feature. This feature attempts to preserve information about both the outermost outline and the inner loops. It is recommended that the Sector Area feature be used in conjunction with Radial Distance.

The Sector Area feature is an array of the areas contained by the outlines within each of the sectors between the radial lines. These areas are also scaled for normalization so that the largest becomes 1.0. Figure 5.1 illustrates the sector area measurements for an example character and the resulting normalized feature vector. The area contained by the outermost loop is directly proportional to the square of the average centre-to-outermost-outline distance mentioned above. Inner loop information is preserved as these loops subtract from the area, giving some indication of their size.

The shape is described not only by the distances and areas but by their position in the respective arrays. By cyclically shifting the elements of the distance and area parts of the array, the character can be recognised regardless of its orientation. Rotational invariance can therefore be achieved at the expense of increased classification time. The original orientation can be preserved, to distinguish characters such as 'n' and 'u' which may be rotations of each other. The method is not truly rotationally invariant since the radial lines will only fall on exactly the same points on the outline if the rotational angle is a multiple of *2π/number of angles used.* An approximately rotationally invariant feature vector is possible however, with invariancy increasing with the number of angles.

The method depends on radial lines intersecting with the image. With unvectorized outlines this is not a problem as even if the reference point is in a line section of the image there will still be an intersection with the outline of that section. With vectorized outlines, problems occur when the reference point falls on a vector: there is no meaningful intersection distance between the reference point and the vector. It was found that the reference point fell on a vector less often using the centre of segment, rather than the centre of gravity, as the reference point.

5.4. Extraction

The feature extraction process operates on a set of top level parent outline loops, together with all their child loops. This set generally represents a single segmented character and is referred to as a *segment*. The first step is to determine a central reference point for the segment using one of the two methods described in section 5.3.

Let *angles* be the number of intervals at which feature measurements are to be taken. This can similarly be thought of as the number of radial lines used. Then *step* is the size of the step taken between each feature measurement, more precisely it is the size of the angle at the reference point between each pair of adjacent radial lines. Let C be the reference point and P and Q be points on an outline in the segment. Let *distance* and *area* be arrays of size *angles*. Each element i of *distance* stores the current greatest distance from C to an outline at bearing $i \times step$ (the bearing of the ith radial line). Each element i of *area* stores the current total area of the segment between radial lines with bearings $i \times step$ and $i \times (step + 1)$, i.e., the current total area of the ith sector. Both arrays are initially zeroized. Note that bearings greater than or equal to 2π will exceed the array boundaries and so the correct array index corresponding to angle θ is $\theta / step \ \% \ angles$, where % is the modulus operator.

The algorithm is as follows:

1. While unexamined outlines of the segment exist, select one. Let *sign* equal +1 if it is an even level loop and -1 if it is an odd level loop.
2. Set P to the first point on the outline and Q to the next point around the outline. Record this initial position of Q.
3. The following measurements are taken:

   θ_P - Bearing of P from C

   θ_Q - Bearing of Q from C

   d_P - Euclidean distance from P to C

   d_Q - Euclidean distance from Q to C

4. If $P = C$ goto step 10.

5. If $Q = C$ and θ_P is a multiple of *step* and d_P is greater than *distance*[θ_P/*step* % *angles*], set *distance*[θ_P/*step* % *angles*] equal to d_P.

6. Using θ_P and θ_Q, determine the range of indices into *distance* and *area* corresponding to radial lines crossed in going from P to Q. Let the first of this range be *start* and the last be *stop*. Determine the direction of movement in going from P to Q. Set the variable *dir* to +1 if the movement is clockwise and to -1 if anticlockwise. This value indicates if the area enclosed by the triangle CPQ, ignoring any loops inside the current outline, is inside (*dir* = +1) or outside (*dir* = -1) the current loop. In combination with *sign* it indicates if the enclosed area is positive or negative (black or white pixels).

7. If *start* = *stop*, a radial line has not been crossed. Add *sign* × *dir* × (area of triangle CPQ) to *area*[*start* % *angles*]. Goto step 10.

8. Let the index i step through *start* to *stop*-1. Let R be a point on the outline where the ith radial line intersects, d_R be the distance from C to R, and θ_R be the bearing of R from C (the bearing of the ith radial line). As the indices are stepped through, the position of R moves along PQ. The example in figure 5.2 shows the positions, R_1 and R_2, of R as it moves from P to Q. The previous position of R is remembered at each stage by advancing either P or Q to that position. Before moving Q its starting position is recorded. For each stage, i, perform steps 8a to 8d:

   8a. If $\theta_P = \theta_Q$, set d_R equal to d_Q. Goto step 8d.

   8b. Set θ_R equal to (*i* % *angles*) × *step*. Find the point R where the radial line with bearing θ_R intersects the line PQ. Calculate d_R.

   8c. If *dir* = +1, the crossed radial lines are being stepped through from P to Q. Add *sign* × *dir* × (area of triangle CPR) to *area*[*i* % *angles*]. Advance P to R; set θ_P equal to θ_R. If *dir* = -1, the crossed radials are being stepped through from Q to P. Add *sign* × *dir* × (area of triangle CQR) to *area*[*i* % *angles*], advance Q to R and set θ_Q equal to θ_R.

   8d. If $d_R >$ *distance*[*i* % *angles*], set *distance*[*i* % *angles*] equal to d_R.

9. Add *sign* × *dir* × (area of triangle CPQ) to *area*[*stop* % *angles*]. Return Q to its starting point recorded in step 8.

10. Advance P to the current position of Q. Advance Q to the next point around the outline from its current position. If Q is not at its initial position (recorded in step 2) goto

step 3, else goto step 1.

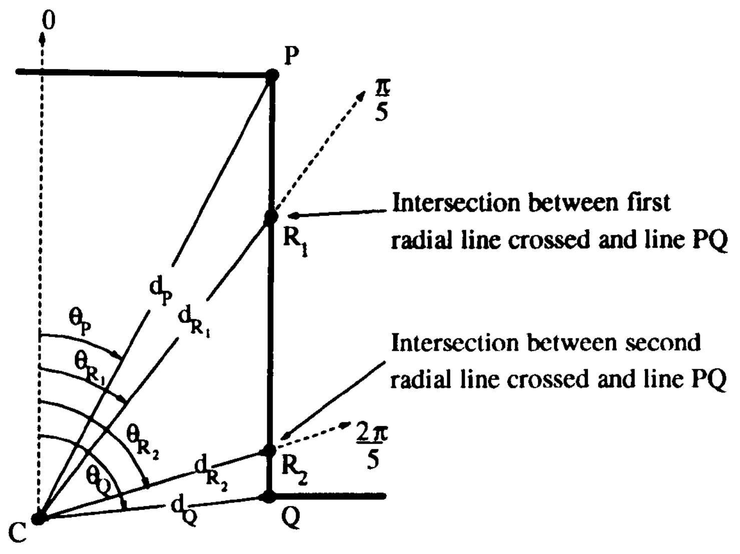


Figure 5.2 Division of the area of triangle CPQ between sectors. The areas of triangles CPR_1, CR_1R_2 and CR_2Q are added to *area*[1], *area*[2] and *area*[3] respectively.

The combined RD/SA feature is simply the concatenation of the *distance* and *area* arrays. The speed of extraction might be increased by pre-calculating sine and tangent values (used in the computation of triangle area and in determining the point R in step 8b) for the radial line angles so they may be quickly found by the algorithm using a lookup table. The algorithm is also much faster if outline coordinates are restricted to integer coordinates, as is the usual form after outlining a binary image. Preprocessing, however, often produces floating point coordinates.

5.5. Results

The RD/SA feature has been tested using the NIST Special Database 3: Handwritten Segmented Characters and the CEDAR CDROM Image Database 1, described in section 2.6. The following data sets have been used:

NIST Digits -	a subset of 3469 pre-segmented handwritten digits (0-9) taken from the NIST database;

NIST Lower -	the complete set of 45313 pre-segmented handwritten lower case characters (a-z) from the NIST database;
NIST Upper -	the complete set of 44951 pre-segmented handwritten upper case characters (A-Z) from the NIST database;
CEDAR Digits -	the complete set of 27688 pre-segmented handwritten digits (0-9) from the combined bi-tonal image training and test sets of the CEDAR database;
CEDAR Lower -	the complete set of 8508 pre-segmented handwritten lower case characters (a-z) from the bi-tonal image training and test sets of the CEDAR database;
CEDAR Upper -	the complete set of 12820 pre-segmented handwritten upper case characters (A-Z) from the bi-tonal image training and test sets of the CEDAR database;

Two types of tests have been conducted, the first to compare the feature's ability to discriminate character classes using different extraction parameters, and the second to evaluate its practical recognition accuracy and ability to generalize to unseen data.

Many statistical measures have been suggested for evaluation of features and their ability to distinguish classes, e.g., discriminant functions [Fuk72] [DH73], Fisher's Criterion [Fis36], probabilistic distance and dependence measures (Bhattacharyya [Bha43], Partick-Fisher [PF69], Chernoff [Che52], Matusita [Mat55], Mahalanobis [Mah36], divergence [NF77]), entropy measures (Vajda, Shannon, Bayesian quadratic, Bayesian cubic) and interclass distance measures (City block, Euclidean, Chebyshev, quadratic, nonlinear). A good overview is given in [Kit86]. Of these, Bayesian quadratic discriminant functions, which assume a normal distribution of the feature measurements, have been chosen for evaluating the RD/SA feature's performance on the character databases. These discriminant functions are applied using a parametric Bayesian quadratic classifier (see section 6.2.1).

The quadratic discriminant gives a probabilistic measure of the ability of the features to discriminate the data sets based on the means and covariance matrices of the sample classes. The error rate of the discriminant is an indication of the amount of overlap of class distributions. In practice the Bayesian classifiers' assumptions about the distribution, means and covariance matrices of the classes are imprecise since they are based on finite sample sets. However, the quadratic classifier's error rate is close to the true Bayes error rate, which is a common measure for evaluating features [SS88].

The quadratic classifier error rate (or alternatively the correct classification rate) is therefore a useful measure for comparing the various methods used in the feature extraction. In the initial tests the quadratic classifier is used in self tests on the complete data sets to evaluate the effectiveness of different normalization factors and reference centres, and to confirm that the addition of the sector area feature and the use of greater numbers of angles will improve the features' performance. Note that the initial self tests are not intended to be an indication of recognition accuracy or generalization ability.

With a view to adaptive segmentation and class-representative feature feedback (see section 5.7 and chapter 8) the learning vector quantization (LVQ) classifier, developed by Kohonen [Koh90a] [Koh90b] [Koh90c], is envisaged for use in the overall system. Since Kohonen compares the accuracy of LVQ with parametric Bayesian classification [Koh90b] they were a logical choice for testing the performance of the feature. In all cases it was found that an LVQ-trained classifier achieved accuracy approximating that of the quadratic classifier. The Bayesian quadratic discriminant therefore provides both a benchmark to compare features and an indication of the potential practical performance using LVQ.

The second set of tests investigates the practical recognition accuracy and generalization ability of the feature. Generalization is the more relevant quality here. Classifier accuracy on the training set is simply a reflection of the ability of the particular type of classifier to learn a data set. The accuracy on unseen data reflects the ability of the feature to capture the distinctive characteristics of the class. Generalization is also, to an extent, dependent on the type of classifier used. However, most classification techniques rely for their generalization on the classes forming clusters in the sample space. Good accuracy on the unseen data therefore indicates good clustering of the classes, which in turn indicates good representation of the distinctive characteristics of the classes.

For the purposes of testing generalization the data sets are divided into a training set, A, and a test set, B. For the NIST data sets the first two-thirds of each class are assigned to set A

Data Set	Samples in Training Set A	Samples in Test Set B
NIST Lower	30203	15110
NIST Upper	29960	14991
NIST Digits	2310	1159
CEDAR Lower	7692	816
CEDAR Upper	11453	1367
CEDAR Digits	24270	3418

Table 5.1 Division of data sets into training and test sets.

and the remaining third to set B. The division of the CEDAR data sets into training and tests sets has already been performed by the providers of the database. The exact number of samples in each set is indicated in table 5.1.

5.5.1. Normalization Factor

Data	Angles used	Feature	Reference centre	Accuracy with normalization factor:	
				Average value	Maximum value
NIST Lower	10	RD only	Segment	68.245%	69.132%
NIST Lower	10	RD only	Gravity	72.730%	74.232%
NIST Lower	10	RD/SA	Segment	77.448%	78.986%
NIST Lower	10	RD/SA	Gravity	81.617%	82.533%
NIST Digits	10	RD only	Gravity	87.187%	89.478%

Table 5.2 Results of varying the normalization factor (scaling the feature vectors so that either the average or the maximum distance/area becomes 1.0).

Two normalization factors were tested for scaling each half of the feature vector: the maximum radial distance (or sector area) for the segment and the average radial distance (or sector area) for the segment. The former factor performed from 1 to 3% better in each of the test cases. Table 5.2 summarizes the recorded accuracy (correct classification rate) on the self tests of the NIST Digits and NIST Lower sets.

After these early experiments it was concluded that the maximum measurement provided the more stable reference value over large data sets. The average value was abandoned as a normalization factor and the maximum value was used in all future testing.

5.5.2. Reference Point

Two methods of selecting a central reference point for the features have been proposed, as described in section 5.3. The *centre of gravity* and *centre of segment* methods have been tested on a variety of test data. Results are summarized in tables 5.3 and 5.4.

On unvectorized data sets the *centre of gravity* method gave greater accuracy in all the test cases. On the combined RD/SA feature the correct classification rate was between 2 and 6% higher than that achieved with the *centre of segment* method. On the Radial Distance feature alone, a more pronounced difference in accuracy was recorded for the upper and lower case sets, with the *centre of gravity* rate 6 to 8% higher. This suggests that its position varies less for similar characters than the *centre of segment* and it therefore provides the more stable, analogous reference point for feature extraction. It is suggested that this method be used when processing unvectorized characters.

On vectorized data, however, the *centre of gravity* performed very poorly as a reference point for reasons discussed in section 5.3. The centre of segment was much more effective by comparison and is therefore recommended as the standard choice of centre point for feature extraction from vectorized characters.

5.5.3. Sector Area

Initial experiments using only the Radial Distance feature vector suggested that improved classification rates might be achieved if some of the interior outline information could be preserved. This led to the development of the Sector Area feature. Table 5.5 shows how

Data	Angles used	Feature	Normalization factor	Accuracy with reference point at:	
				Centre of Segment	Centre of Gravity
NIST Lower	10	RD only	Average	68.245%	72.730%
NIST Lower	10	RD only	Maximum	69.132%	74.232%
NIST Lower	10	RD/SA	Average	77.448%	81.617%
NIST Lower	10	RD/SA	Maximum	78.986%	82.533%
CEDAR Upper A	10	RD/SA	Maximum	86.044%	87.886%
NIST Upper	10	RD/SA	Maximum	88.232%	90.180%
NIST Digits	10	RD only	Maximum	89.132%	89.478%
NIST Digits	10	RD/SA	Maximum	95.244%	96.397%

Table 5.3 Results of varying the reference point for unvectorized data sets. Accuracy is the correct classification rate of the quadratic discriminant classifier on the self test of the data set.

Data	Angles used	Feature	Normalization factor	Accuracy with reference point at:	
				Centre of Segment	Centre of Gravity
NIST Digits	10	RD only	Maximum	81.781%	58.943%
CEDAR Digits B	10	RD only	Maximum	75.834%	51.726%

Table 5.4 Results of varying the reference point for vectorized data sets.

combining the Sector Area feature with the original Radial Distance vector gave rise to a significant increase in self test accuracy (an improvement of between 8 and 15% on the data sets tested).

Data	Angles used	Normalization factor	Reference centre	Accuracy with feature:	
				RD only	RD/SA
NIST Lower	10	Average	Segment	68.245%	77.448%
NIST Lower	10	Maximum	Segment	69.132%	78.986%
NIST Lower	10	Average	Gravity	72.730%	81.617%
NIST Lower	10	Maximum	Gravity	74.232%	82.533%
NIST Digits	10	Maximum	Segment	89.132%	95.244%
NIST Digits	10	Maximum	Gravity	89.478%	96.397%

Table 5.5 Comparison of using the Radial Distance feature alone and using the combined Radial Distance/Sector Area feature.

Introducing the Sector Area feature involves a slight decrease in speed but no change to the order of complexity of the algorithm. The increase in computation time is negligible compared to the increase in accuracy.

5.5.4. Angles Used

Data	Feature	Normalization factor	Reference centre	Angles used	Accuracy
NIST Lower	RD/SA	Maximum	Gravity	8	80.939%
NIST Lower	RD/SA	Maximum	Gravity	10	82.533%
NIST Lower	RD/SA	Maximum	Gravity	15	83.713%
NIST Lower	RD/SA	Maximum	Gravity	30	84.742%

Table 5.6 Results of varying the number of angles (radial lines) used for the NIST Lower data set. Accuracy is the correct classification rate of the quadratic discriminant classifier on the self test of the data set.

Ten angles were used for the majority of tests. It was found that this number of radial lines and sectors was quick to compute and provided reasonable accuracy for use in the development of a test recognition system. It was expected that increasing the number of angles would increase recognition accuracy at the expense of speed. Determining an acceptable number for a practical system was not an objective of this research. However, a brief comparison was made of classification rates on the unvectorized NIST Lower data set for a selection of vector sizes.

Clearly the greater the number of angles used the greater the accuracy. The size of feature vectors slows processing both at the feature extraction stage and at the subsequent classification, learning or reconstruction stages. The feature extraction may be computed in parallel. Speed restrictions on the practical sizes of feature vectors are therefore more likely to result from the limitations of the subsequent classification processes.

5.5.5. Practical Recognition Accuracy

Practical tests of the RD/SA feature's recognition accuracy and generalization ability have been conducted using the quadratic discriminant classifier and the learning vector quantization (LVQ) classifier. For reasons discussed in later chapters, LVQ has properties desirable in the envisaged hierarchical classification system. It should be noted however, that greater accuracy on both the training and test sets might be achieved using more powerful classifiers that were deemed unsuitable for the hierarchical system, e.g., multilayer perceptrons. The quadratic classifier is used in these tests as it provides a good guide to the feature's generalization ability. As pointed out by Kwan *et al.* [KPS79], the use of a simple classification scheme emphasizes the features over the classifier.

The implemented system is designed to aid future testing of the mechanisms for classification, contextual processing and feedback in the hierarchical character recognizer proposed in chapter 8. It places speed above accuracy as this will facilitate fast testing of those mechanisms. Therefore only a small number of angles — ten — are used for feature measurement. In a fully working system more angles could be used, and as a result more time spent in training, to provide greater accuracy. It is found, however, that ten angles give very reasonable accuracy and generalization on the data sets.

Training Data	Test Data	Classifier	Accuracy on:	
			Training Data	Test Data
NIST Lower	CEDAR Lower	Q.D.	82.53%	70.31%
NIST Upper	CEDAR Upper	Q.D.	90.18%	79.09%
NIST Digits	CEDAR Digits	Q.D.	96.40%	87.07%
NIST Lower	CEDAR Lower	LVQ (364)	80.39%	71.06%
NIST Upper	CEDAR Upper	LVQ (364)	91.20%	76.84%
NIST Digits	CEDAR Digits	LVQ (77)	97.20%	86.48%
CEDAR Lower A	CEDAR Lower B	Q.D.	87.69%	84.79%
CEDAR Upper A	CEDAR Upper B	Q.D.	87.89%	85.79%
CEDAR Digits A	CEDAR Digits B	Q.D.	92.55%	90.11%
NIST Lower A	NIST Lower B	Q.D.	81.98%	83.49%
NIST Upper A	NIST Upper B	Q.D.	90.11%	90.37%
NIST Digits A	NIST Digits B	Q.D.	96.88%	92.24%
NIST Lower A	NIST Lower B	LVQ (364)	84.32%	87.76%
NIST Upper A	NIST Upper B	LVQ (364)	93.45%	89.81%
NIST Digits A	NIST Digits B	LVQ (77)	98.01%	89.47%

Table 5.7 Accuracy of the quadratic discriminant and learning vector quantization (LVQ) classifiers on the NIST and CEDAR data sets. The number in parentheses in the Classifier column indicates the number of codebook vectors used by the LVQ classifier.

As a result of the initial tests in sections 5.5.1-5.5.4 it was decided that the maximum distance or area measurement would be used as the normalization factor and that the reference point used would be the centre of gravity, except when processing vectorized images in which case the centre of segment is used. The Sector Area feature would be used in each case, except for vectorized images where its use is precluded.

Generalization results on the CEDAR data using classifiers trained on the NIST data are markedly poorer than the results using the same data source for both training and testing. This is mainly because the CEDAR characters have been edited to only consist of one connected component. They are therefore often lacking parts of characters, such as the dots on 'i's and 'j's, which are present in the NIST characters. This makes the generalization task much more difficult. The tests using the same source for training and test data provide a fairer indication of the feature's generalization ability in a practical application, where it is likely that the same data gathering procedure will be used in both training and actual use.

Accuracy and generalization using the same source for testing and training are encouraging. The LVQ results approximate those of the quadratic discriminant classifier. The generalization on the NIST data in the implemented LVQ-based system is in the high eighties for each category of characters. This is as good as can be expected for recognition of upper and lower case letters in the absence of context. In the case of digits, where context is less likely to be able to correct errors, the generalization would need to be improved for use in a practical application. More angles must be used on the feature extraction.

In the implemented system the LVQ classifier is modified to produce a candidate set of most likely classes rather than a winner-take-all result. Table 5.8 gives an indication of the percentage of cases where the correct class will be included in the candidate set. It records the percentage of cases where the correct class appears in the top three closest classes to codebook vectors, i.e., the three most likely classes predicted by the LVQ classifier.

The "top three" results on upper and lower case letters suggest that contextual processing of the candidate sets could potentially produce high levels of recognition accuracy using only a small number of angles in the feature extraction. Although the results on digits are high, there may be less potential for contextual disambiguation of digits in the particular application. The "top three" results might therefore be overly optimistic for general digit recognition; such high results could only be expected in form processing where numerals on forms could be matched against other data, e.g., courtesy amounts on cheques could be matched

Training Data	Test Data	Classifier	Top Three Accuracy on:	
			Training Data	Test Data
NIST Lower	CEDAR Lower	LVQ (364)	91.04%	88.66%
NIST Upper	CEDAR Upper	LVQ (364)	96.51%	90.54%
NIST Digits	CEDAR Digits	LVQ (77)	99.11%	96.50%
NIST Lower A	NIST Lower B	LVQ (364)	92.69%	93.61%
NIST Upper A	NIST Upper B	LVQ (364)	96.66%	96.88%
NIST Digits A	NIST Digits B	LVQ (77)	99.05%	97.58%

Table 5.8 "Top three" accuracy of the learning vector quantization (LVQ) classifier on the NIST and CEDAR data sets. Accuracy results are the percentage of cases where the correct class is in the top three most likely classes calculated by the classifier.

against the legal amounts.

5.5.6. Further Testing

The RD/SA feature has been tested commercially by Seel, a company developing an automated form reader for SKY Television applications. Seel used a database of 20,000 training samples (isolated handwritten digits) and 2,500 test samples, taken from real application forms. The numerals were slant normalized before the RD/SA features were extracted.

Rather than learning vector quantization, a more powerful back-propagation classifier was used for training and testing (see section 6.4.3). The classifier produced a certainty measure for candidate output classes. Where the difference in certainty between the two most likely candidates fell under a certain threshold the case was rejected, otherwise the first most likely candidate was output as the classification. The incorporation of rejection into the classification aims to reduce the error rate by detecting and avoiding potentially ambiguous cases. It is common in practical applications for rejection to be preferred to error — rejected cases can easily be processed by hand.

Seel reported that the RD/SA-based system performed comparably to the best of the commercially available OCR packages currently used by the company, equaling the lowest rejection rate within 0.5%. This result clearly shows accuracy of the RD/SA feature and its robustness when dealing with real-world data.

5.6. Comparison of the Feature on Outlines and Vectors

It has been argued in chapter 4 that preprocessing images in an attempt to make them fit human abstractions of character shape can often lead to increased errors in the data set and loss of detail. The Radial Distance feature has been used to compare unpreprocessed outlines with their preprocessed (vectorized and smoothed) counterparts. The stroke-based vectorization method described in chapter 3 has been used for this purpose.

This comparison is, in effect, a test of the vectorization method's ability to preserve geometric information from the outline. In addition it is a test of the RD feature's effectiveness on vectorized characters. However, the vectorization is intended to approximate the outlines at a slightly higher level of abstraction, using strokes, or parts of strokes, as primitives for structural classification. In this classification approach, the structural relationships of the character are considered more important than the detail of the shape. Section 4.3.1 discusses the use of vectorization for structural primitive extraction and section 6.3 describes structural classification. In this section it should be noted that the potential usefulness of this structural representation is not taken into account by the RD feature so this experiment does not fairly compare the worth of minimal preprocessing with the traditional preprocessing approach. However, it does show that the vectorization method used either loses information about the basic geometric shape of the outlines or introduces new errors, or both.

Three data sets were compared in both unvectorized and vectorized forms. Ten angles were used and only the Radial Distance feature was measured (it is assumed that area information is lost in the vectorized representation). As shown in section 5.5.2 the *centre of gravity* method of selecting a reference centre was ineffective and so the *centre of segment* is used instead on vectorized characters. This method is therefore also used on the outlines for the

Training Data	Test Data	Accuracy on Train Set		Accuracy on Test Set	
		Outlines	Vectors	Outlines	Vectors
NIST Lower	CEDAR Lower	69.69%	60.88%	58.69%	49.07%
NIST Lower A	NIST Lower B	68.10%	59.97%	70.07%	62.06%
CEDAR Lower A	CEDAR Lower B	74.23%	63.88%	74.38%	62.32%
CEDAR Upper A	CEDAR Upper B	71.50%	65.90%	70.26%	63.97%
NIST Digits	CEDAR Digits	89.13%	81.78%	78.64%	72.61%
NIST Digits A	NIST Digits B	89.52%	83.33%	86.63%	77.83%
CEDAR Digits A	CEDAR Digits B	83.30%	78.40%	81.89%	75.92%

Table 5.9 Comparison of the Radial Distance feature on vectorized and unvectorized (outline) data sets.

purpose of comparison, although the *centre of gravity* is generally more accurate. The maximum distance measured is used as the normalization factor. All tests use the quadratic discriminant classifier.

The results show that the vectorization and preprocessing employed causes a drop in test set accuracy of between 7 and 16% from the unvectorized data sets. Clearly some of the distinguishing geometric detail of the characters has been lost in the process, or alternatively errors have been introduced into the shape.

5.7. Adaptive Segmentation

As described in section 2.4, a recursive approach to segmentation has been employed by several researchers. These attempts, used predominantly on machine-printed text, perform iterative segmentation followed by classification. When the certainty of classification falls below an acceptance level, segmentation is corrected by selecting a new, predetermined, point at which to split the characters, usually using a vertical straight line. The classifier output is not used except to indicate acceptance or rejection of the classification.

A more powerful approach is proposed for cursive script segmentation. Since accurate segmentation of characters is impossible without some knowledge of what they are, an adaptive approach is proposed which uses feedback from the classification and contextual processes to direct changes in segmentation, based on expectation of what the characters are.

After an initial blind segmentation, followed by a first attempt at classification, the system may have the beginnings of a picture of the image at a higher level of abstraction than the raw outlines and feature vectors. The initial results of the classification, in particular the certainty of individual character classifications, are a reflection of the accuracy of the segmentation and are used accordingly to correct that segmentation. Small adjustments may be made to the segment boundaries where the certainty of classification is high, to make those boundaries more precise. Where the certainty is poor, larger adjustments might be made on the assumption that the initial segmentation is completely wrong.

This classification and adaptation process iterates until all character classifications are at a reasonable certainty or such a certainty seems unattainable. Chapter 8 discusses in more depth the possibilities of using classifier feedback to enhance recognition. Let us now consider how the RD/SA features can be used to direct the adaptation of segmentation.

It is assumed that the classifier, in conjunction with contextual processes, can provide a feature vector representing the expected class of a segment, i.e., after passing a feature vector to the classifier it is expected that the classifier will decide on a class for that vector and pass back a feature vector representative of that class. The use of feedback from contextual classifiers to decide on contextually valid character classes is discussed in much greater detail in chapter 8. The geometric correspondence of the first part of the feature vector to the outline it represents allows the actual outline of the segment to be compared geometrically to the expected outline of the character (assuming the classification is correct).

A system of over-segmentation of cursive script has been devised which divides the outline into many small sections. A word then consists of a number of segments, each of which consists of several of these small sections. Where the expected character outline differs from the actual outline of the target segment, that segment can be adjusted by moving small sections either to or from the adjacent segments. The difference in the expected and actual

Radial Distance feature value in a given direction gives an approximation of the change required in the segmentation.

The small sections in that direction, in both the target segment and its adjacent segment, are identified and their sizes determined. The difference in feature value, scaled according to the largest radial distance (to account for the normalization), indicates the size of re-segmentation required. One or more small sections are moved accordingly, from one segment to the other, so that the new feature value in that direction will approximate the expected value. Note that it can only be an approximation as the shift of sections will affect the central reference point. The Sector Area feature might also be used to judge the required movement of small sections: the difference in expected and actual areas of the sectors either side of the radial line could be used to limit the addition or subtraction of sections from a segment.

Once each direction has been checked and any desired segmentation adjustments have been made all feature vectors will need to be recalculated. The feedforward and feedback classification processes are then repeated until, through a process of relaxation, the segmentation settles on a solution where, ideally, no further adjustments are required.

Testing of the adaptive segmentation proposal requires an implementation of the contextual feedback architecture of chapter 8 which is beyond the time constraints of this work. Preliminary work has developed a method for dividing cursive script into small sections. It is found that a large number of divisions are required to cover all possible segmentations.

It is assumed that the orientation of the cursive word is known. Segmentation into small sections is based on finding pairs of possible segmentation points on the top and bottom edges of the outline and splitting the outline along the line joining the points. Possible segmentation points are placed at peaks and troughs on the outline; specifically, at local minima on the top edge and local maxima on the bottom edge. (Minima and maxima at the very end of an outline are ignored.) This applies only to outermost (level 0) outlines and large outlines immediately inside them (level 1 loops) whose area exceeds some constant, DOT_AREA. Anything smaller is ignored. Note that for level 1 loops, what appear to be top edge points (on the top edge of the loop) are termed bottom edge points, and vice versa, since they are on the bottom of the black pixel region.

For each possible segmentation point, between one and three pairs are found. For a top edge segmentation point, *A*, on a level 0 loop, the pair points are:

B - a point on the bottom edge directly below the first point (i.e., the segmentation is by a vertical line);

C - the first possible segmentation point on the bottom edge of the level 0 loop, immediately to the left of *B*;

D - the first possible segmentation point on the bottom edge of the level 0 loop, immediately to the right of *B*.

C and *D* may not always exist. For a bottom edge segmentation point, the pairs are the same but are on the top edge. For level 1 loops only point *B* is used.

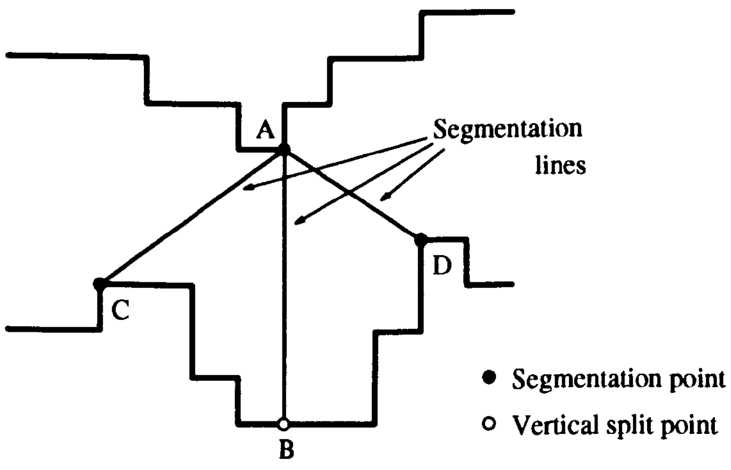


Figure 5.3 Pair points for segmentation point *A*.

To reduce the number of segmentation points the vertical (*A*-*B*) splits are tested before any segmentation takes place to determine the area of the two sections that would result if the split was made. If either of these areas is less than or equal to DOT_AREA the segmentation point is removed on the assumption that that area is too small to be a character. Note that if *A* is on a level 0 loop and *B* is on a level 1 loop, or vice versa, there will be no division of the area as a result of the split. In this case the segmentation point is always kept.

Where the line between a pair is entirely contained within the outline (the black pixel area), and the top edge point is above the bottom edge point, the outline is divided along that line. Where segmentation lines for different pairs cross, the diagonal splits, *A*-*C* and *A*-*D*, have priority over the vertical splits. After the diagonal splits have been made, the vertical ones are recalculated if necessary, so that *B* now falls directly below *A* on the line of the diagonal split.

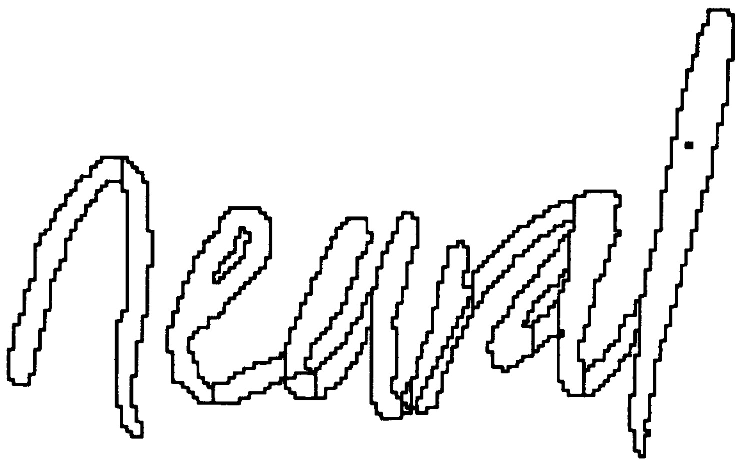


Figure 5.4 Typical division of a cursive word, "neural," into small sections. Integer coordinates and orthogonal edges have been preserved through the use of Bresenham's line drawing algorithm.

Once this division is complete many small sections remain. Figure 5.4 illustrates a typical division of a cursive word. To retain integer coordinates and orthogonal edges for the small outlines, Bresenham's line drawing algorithm [Bre65] has been used to produce the coordinates of points along the diagonal splits.

Future work must develop a method for the initial assignment of small sections to character segments. Clearly there are a large number of combinations of possible assignments and more work needs to be done to try to reduce the number of sections involved. Fast, parallel feature extraction and subsequent classification is essential to the adaptive segmentation problem. An exhaustive search of all the combinations would be prohibitively time consuming; direction of the search by intelligent feedback is necessary to make the system fast enough for practical use (see chapter 8).

5.8. Extensions

Several possibilities have been considered for future work on developing the RD/SA feature. Improvements to the basic feature might look at more accurately measuring the inner loops or interior points on the outer loop. When a contour bends back on itself only the outer edge is measured by the feature. Points on the inside edge, closer to the reference centre, might also be of importance to the discrimination of character classes. An additional vector might therefore be added which measures the second furthest outline point from the reference centre for each radial line. Inner loops would also be considered by this feature. The feature might be extended still further to incorporate vectors of the third furthest distances, or the fourth, and so on. It is unlikely that measurements beyond the third furthest distances would be worthwhile; such features would be zero vectors for most characters, i.e., the outlines do not cross the radial lines more than three times.

The main desirable quality of the feature that has not been explored in this research is the speed of computation. While improvements might be made to the basic extraction algorithm (removing floating point arithmetic from the process wherever possible) the current method is reasonably efficient. However, in a system using the suggested method of adaptive segmentation, the possibility exists for a significant reduction in extraction time.

In the segmentation system described in section 5.7, feature extraction may be performed many times before a character outline is segmented correctly. However, much of the segment will be the same each time. A highly desirable property for a feature would be the ability to recalculate it rapidly from its previous value, i.e., calculate the change in the feature from the change in segmentation.

The RD/SA feature does not have this property. It cannot be calculated in an incremental way, as small sections are added to or subtracted from the previous segment, because a slight change in area changes the position of the reference centre. The feature needs to be completely recalculated every time. Future development of the feature must explore other forms of reference points, and corresponding feature measurements, that can be recalculated directly from the change in segmentation.

One possibility is to use a vertical line through the centre of a character segment as a reference line. Feature measurements would be the horizontal distances from the line to the furthest point on the outline, measured at a variable number of equidistant points along the reference line. The area of the image between adjacent horizontal lines could be measured as an equivalent to the Sector Area feature. These measurements might be facilitated by using a run-length representation of the image rather than its outline. Segmentation would be by simple vertical lines; re-segmentation would shift vertical strips from one segment to the next. The change in position of the reference line could be determined from the change in strips in each segment. Where strips are added to a segment, the change in features could be calculated using only the added strips. The difficulty of this development is that size normalization could not be handled as easily. Accuracy is likely to be comparable to that of the RD/SA feature. The limitations of this variation are that simple vertical segmentation may not be sufficient for correct division of characters, and that the property of rotational invariance is lost.

5.9. Evaluation

The use of a central reference point, feature normalization and the possibility of cyclic shifting of the feature vectors makes the RD/SA feature invariant to position, size and rotation respectively. Its accuracy can be adjusted by varying the number of angles used in the extraction. The results of sections 5.5 and 5.6 show that high rates of correct classification can be achieved on handwritten digit and character data using only a small number of angles, and using relatively simple classification methods, both on the training samples and on unseen test data. This demonstrates the ability of the feature to capture the characteristics of classes from their outermost outlines.

The main advantage of the RD/SA feature is its ability to cope with broken outlines or multi-outline characters. Its measurements are not limited to a single closed loop and so recognition does not have to be based on the largest outline in a segment, or rely on structural rules to relate, for example, the dot of an 'i' with its stalk. This robustness is an essential property for any practical character recognition application and a significant improvement over many other features.

Feature extraction is reasonably fast and the independent nature of the measurements makes the process suitable for a very fast parallel implementation.

The feature vector is of fixed length, making it practical for matching against other feature vectors. Since it is, in part, a geometric description of the outline, it is possible, to an extent, to reconstruct the original outline from the feature vector. This is a useful capability for a recognition system envisaged to use contextual feedback (see chapter 8) and adaptive segmentation (section 5.7), and was a primary consideration in developing the RD/SA feature.

The RD/SA feature is inherently more tolerant to noise than most other outline-based features. Unlike contour following descriptors, such as Dinstein's features, polygonal approximations and, to a lesser extent, outline-based series expansions, each of the RD/SA feature elements is entirely independent. Thus noise in one sector, or along one radial line, will only affect one element of the feature vector. Additionally, the effect of noise is usually minimal.

Small distortions in the outline, which can produce large variations in outline following features, may cause no variation at all in the Radial Distance feature if no radial line intersects the distortion. In cases where a radial line does pass through a small noisy area, the proportional change in measurement should not, in most cases, be large enough to affect the overall characterization of the shape. Areas of noise along a radial line that are separate from the actual character outline may cause large changes in that Radial Distance. It is expected that accurate grey level thresholding can remove very small spots of background noise so only the much rarer, large noise components will remain to cause these large changes. However, removing such large areas from consideration is a task for the segmentation process, not the feature extractor. The only major corruption of Radial Distance features occurs when a radial line passes cleanly through a break in an outline. The likelihood of this occurrence increases with the number of radial lines used and hence the accuracy of the representation. It is hoped that the increased accuracy will compensate for some of the occasional false measurements.

The Sector Area features are only intended as a rough guide to area distribution which enhances the Radial Distance features, rather than being highly accurate discriminant features themselves. The characterization of shape by the combined RD/SA feature is therefore unlikely to be affected by small changes in Sector Area, such as spurs on outlines and spots of background noise. Large changes in area may be caused by large noise components (a segmentation problem) or broken outlines.

The RD/SA feature is therefore highly tolerant to noise, though occasionally susceptible to breaks in outlines. Breaks due to hasty writing commonly occur at similar points on the characters so this problem may be partly addressed by using a large training set which reflects these common break points. Style variation must also be learned using a large sample set. The results show that classification systems exhibiting high tolerance to style, slant, noise and breaks can be built using the RD/SA feature in combination with representative training databases.

5.10. Conclusions

A robust feature for outline representation has been developed which is invariant to position and size and approximately invariant to rotation. It is fast to extract and is highly tolerant to noise compared to other outline-based features. Broken and multi-part images are easily handled by the RD/SA feature.

Original outlines may be partially reconstructed from the feature. An adaptive recursive segmentation strategy has been proposed which uses this property to direct more powerful segmentation of text. Future work on the adaptive segmentation problem must aim to integrate the segmentation and feature extraction processes. The measurement approach taken by the RD/SA feature might be developed to facilitate this integration.

It has been shown that the Radial Distance/Sector Area feature captures the distinctive characteristics of character classes from the outer edges of outlines; it is accurate and generalizes well when trained on representative sample sets. High recognition rates have been achieved at reasonable speeds on several character databases.

5.11. Nomenclature

%	Modulus operator.
θ_P	Bearing of *P* from *C*.
θ_Q	Bearing of *Q* from *C*.
θ_R	Bearing of *R* from *C*.
angles 18	Number of intervals in which feature measurments are taken.
area	Array of sector area measurements.
C	Central reference point.
d_P	Euclidean distance from *P* to *C*.
d_Q	Euclidean distance from *Q* to *C*.
d_R	Euclidean distance from *R* to *C*.
dir	Direction of movement around outline (clockwise/anticlockwise).
distance	Array of radial distance measurements.
DOT_AREA	Lower limit on area of outlines to be considered for segmentation.
p	Unit pixel width.
P	Starting point of the current section of outline.
Q	Stoping point of the current section of outline.
R	Point on outline where a radial line intersects *PQ*.
sign	Indicates even or odd level outline loop.
start	First potential index into feature arrays for line from *P* to *Q*.
step	Size of angle step between each feature measurement.
stop	Last potential index into feature arrays for line from *P* to *Q*.

Chapter 6

Character Classification and Contextual Processing

6.1. Summary

Classification is the process by which a set of input variables are mapped to a set of output categories or classes. In the case of character classification, the input variables are usually feature strings, vectors or sets of the kind described in section 2.5, and the output classes are characters, e.g., '0', '1',..., '9', 'a', 'b',..., 'z'. The process is slightly more complicated if we wish to output a *candidate set* of possible classes, rather than a single winner-take-all classification.

This chapter reviews the field of classification, as it applies to character recognition. Character classification techniques can be broadly divided into *statistical*, *structural* (including *syntactic*) and *neural* approaches [Sch92]. Sections 6.2 to 6.4 describe these three approaches and examples of those types of classifiers.

An important research area very closely related to classification is that of contextual processing. This is another large field, concerned with the use of external knowledge to correct or disambiguate classifications and to provide a higher level of understanding of the input. It is particularly relevant to character recognition where characters cannot be accurately or unambiguously classified in isolation and require an examination of a wide context and a greater understanding of the text. Section 6.5 discusses the need for contextual processing and section 6.6 surveys the methods that have been applied to character recognition. Conclusions are made in section 6.7 and section 6.8 describes the notation and symbols used in this chapter.

6.2. Statistical Classification

Statistical (or decision theoretic) character classification is a feature matching technique. Feature measurements (see section 2.5) form a vector in the feature space. The goal of the statistical classifier is to determine which class a feature vector belongs to. This is essentially a task of finding *decision boundaries* (or *class boundaries*) in the feature space which delimit the regions corresponding to each class.

This category of classifier has a firm basis in statistical theory. This theory, as it applies to statistical pattern recognition, is covered in detail in many textbooks, e.g, Duda and Hart [DH73], Bow [Bow84] and Schalkoff [Sch92].

There are four main types of statistical classifiers: *parametric*, *nonparametric*, *unsupervised* and *fuzzy decision-making*. The following four sections describe each type of statistical classifier as it relates to off-line character recognition.

6.2.1. Parametric Classification

Parametric classifiers can be used when the parametric probability distributions of the classes and the *a priori* probabilities of those classes are known. These parameter values allow *decision functions* to be designed which perform classification according to Bayesian decision theory. Each class has an associated *discriminant function* (introduced by Fisher [Fis36]) which is computed for each input vector. The decision function determines the output class to which the input vector is assigned according to these discriminant functions. The output class is the one associated with the discriminant function producing the maximum (or alternatively the minimum) value.

Bayesian classifiers use Bayes' rule as a basis for their decision functions. A vector of input measurements, $\mathbf{x}$, can be taken from any given object, and that object can be assigned to the class with the highest conditional probability (the probability of the class given $\mathbf{x}$) according to Bayes' rule:

$$P(C_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C_i)P(C_i)}{\sum_{all\ i} P(\mathbf{x} \mid C_i)P(C_i)}$$

where C_i is class i. It is also possible to calculate the risk involved for each value of $\mathbf{x}$ (the probability of assigning the object to the wrong class) and the *Bayes error* (the minimum achievable error), which is the expectation of the risk function over the whole feature space. It can be shown (see Duda and Hart [DH73]) that the Bayes decision rule minimizes the Bayes error.

The difficulty with the practical application of this method is that the class conditional densities ($P(\mathbf{x} \mid C_i)$) or the *a priori* class probabilities ($P(C_i)$) may not be known. This is the case for most classification problems including character recognition. The design of parametric classifiers requires estimation of these probabilities and density functions.

In cases where the functional forms of the class conditional probability density functions are known, the task is to estimate the parameters of the particular form of function. The two most popular forms of parametric classifier are the linear and quadratic classifiers. These both assume a Gaussian (also called normal) distribution of the class conditional densities and are designed by estimating the parameters of the Gaussian functions. Gaussian density functions are popular because they can be completely defined by a relatively small number of parameters, are continuous and defined over the whole feature space and produce simple decision functions (at most quadratic functions of $\mathbf{x}$) [KD]. Also, variables in real life problems are very often normally distributed. For these reasons, the quadratic classifiers which arise from the Gaussian distribution assumption are commonly used even when the density function is not known to be Gaussian [Fuk86a].

Assuming a Gaussian distribution of the variables making up $\mathbf{x}$, the form of the class conditional probability density function is:

$$P(\mathbf{x} \mid C_i) = \frac{1}{(2\pi)^{n/2} \mid \Gamma_i \mid^{1/2}} \exp[-\frac{1}{2}(\mathbf{x} - \mu_i)^t \Gamma_i^{-1} (\mathbf{x} - \mu_i)]$$

where μ_i is the mean vector of class i and Γ_i is the covariance matrix of class i. The means and covariance matrices are the parameters that must be estimated.

Substituting this function into Bayes' rule leads to the decision rule:

Assign to class i if

$$d_i(\mathbf{x}) - \ln(P(C_i)) \; < \; d_j(\mathbf{x}) - \ln(P(C_j)) \quad \forall j \neq i$$

where $d_i(\mathbf{x})$ is the discriminant function for class i:

$$d_i(\mathbf{x}) = \ln \mid \Gamma_i \mid + (\mathbf{x} - \mu_i)^t \Gamma_i^{-1} (\mathbf{x} - \mu_i)$$

James [Jam85] gives a good explanation of the derivation of this decision function.

The *parametric quadratic* classifier (sometimes called *quadratic discriminant* classifier) is comparatively powerful as it forms quadratic decision boundaries. The feature space is partitioned by hyperquadric decision surfaces which can take any of the general forms (hyperplanes, hyperspheres, hyperellipsoids, hyperparaboloids or hyperhyperboloids). The much simpler *parametric linear* classifier results when the covariance matrices of the classes are identical. The additional assumption that these matrices are identical is sometimes made in order to simplify the decision rules. The discriminant functions of the linear classifier perform linear divisions of the sample space (division by hyperplanes). The basic idea is that points on one side of the division are assigned to one class and those on the other side are assigned to another, although in practice it is slightly more complicated.

For character recognition the simple decision boundaries of the linear classifier are unlikely to be able to discriminate character classes. A linear classifier was tested on the Radial Distance/Sector Area features of chapter 5 and gave a very high error rate, even on the self test. The quadratic classifier performed with much greater accuracy (see the results in sections 5.5 and 5.6).

The parameters of the classifiers' class conditional density functions are estimated from a sample set of the population, usually called a *training set*, where each sample has a label indicating the class to which the sample belongs. The training set is taken to be representative of the whole population. The parameters of the training set density function can be determined and then these are taken to be the parameters of the actual density function.

There are three main strategies for parameter estimation: point, interval and Bayesian estimation. In point estimation the parameter vector is taken to be an unknown constant. The constant is determined from a function of the parameter vector and the training set. For example, the maximum-likelihood estimation uses a dummy parameter vector and computes the probability of sampling the training set given that vector; the dummy vector that produces the greatest probability is selected as the parameter estimation. Interval estimation determines a range within which the true parameter vector falls and a confidence factor for that range. Bayesian estimation assumes the parameter vector is a random variable with a known *a priori* probability density function of its own. The *a posteriori* density function of the parameter vector can be determined by examination of the training set. A sharp peak in the function is taken to indicate the true value of the parameter vector. Detailed descriptions of these estimation techniques can be found in most statistics textbooks; comprehensive expositions can be found in Beaumont [Bea80] and Therrien [The89]. Point and Bayesian estimation are generally preferred over interval estimation as they require fewer training samples to reliably represent the population [KD].

The practical difficulty with parametric classifiers, and with statistical classification in general, is that for real world problems with many feature variables, the number of samples required to accurately estimate the density function is very large. Often the required training set is too large to be collected and the parameters have to be estimated from an insufficient number of samples. The resulting inaccuracy in the parameter estimation causes inaccuracy in the classification. However, sample size is not as much of a problem for parametric classifiers as it is for nonparametric ones. Accurate parametric classifiers are easier to design and so they are preferred over nonparametric classifiers whenever they are applicable.

6.2.2. Nonparametric Classification

Where the form of the underlying class conditional probability density function cannot be assumed, parametric classification cannot be applied. There are two approaches to nonparametric classification. The first is to estimate the density function from a training set without simplifying the function to a set of parameters. The second is to avoid the determination of the class conditional density functions altogether and produce a classifier directly from a labelled training set.

Nonparametric estimation of the density function of a class is achieved by representing the function at each point in feature space as the sum of contributions from all the training samples belonging to that class. Each sample contributes to the function according to a density kernel function, for example, a rectangular function, truncated squared-cosine function, or a potential function (Bashkirov *et al.* [BBM64]) where the sample is treated as a charged particle causing an electrical potential in the feature space. Therrien [The89] gives a comprehensive survey of these techniques. Nonparametric density estimation requires a large number of samples to reliably represent the true density function. It is preferable to use parametric estimation wherever possible.

The most common methods of nonparametric classification, without estimation of the class conditional densities, are the *k-nearest-neighbour, nonparametric linear* and *piecewise linear* classifiers (also called *linear machines*). There are also *nonparametric quadratic* and *piecewise quadratic* classifiers though these are less commonly used.

The *nearest neighbour classifier* (Cover and Hart [CH67]) stores the set of labelled training samples and classifies a test sample by assigning it to the class of the closest training sample in feature space. Closeness is determined by a distance metric, usually the Euclidean distance, though for well clustered data the Mahalanobis distance is often more effective [But93]. The *k*-nearest-neighbour classifier is an extension of this which finds the classes of the *k* closest training samples and assigns the test sample to the class which occurs most frequently. The accuracy of the nearest-neighbour increases with the number of training samples, but more samples also increase the computation time.

It can be proven that the error rate of the nearest-neighbour classifier is bounded by twice the Bayes error rate (see Duda and Hart [DH73]). However, in many applications parametric classifiers can be found which produce lower error rates [Fuk86a]. Nearest-neighbour classifiers are slow in operation as they must compare the test sample to all the training samples. For a practical application the number of training samples needed for accurate representation of the classes will be very large and hence the classifiers will be extremely slow. Nearest-neighbour classifiers are therefore rarely useful for practical problems such as character recognition. Two strategies have been attempted to increase the speed of classification. One is to reduce the computation time of the search algorithm, for example, by using search trees (Bentley and Friedman [BF79]). The second is to reduce the number of training samples required. Methods for this include taking out training samples which are mis-classified by the rest of the set (Devijver and Kittler [DK82]), and storing only the training samples at the boundaries of classes (Hart [Har68]). A more effective sample reduction method is vector quantization which will be described later in section 6.4.4.

Whereas parametric linear and quadratic classifiers assume the forms of the underlying probability distributions and use the sample set to estimate the parameters of those distributions, nonparametric linear and quadratic classifiers assume the forms of their discriminant functions and use the sample set to estimate the parameters of those functions.

The simple linear and quadratic classifiers have only one discriminant function and so can only discriminate two classes. Multi-class problems can be tackled using multiple instances of the two-class classifiers. This is a common approach but it can allow areas of the input space for which the classification is undefined [DH73]. A better approach is to use the more powerful piecewise classifiers. The piecewise versions are the general forms which have one discriminant function for each class and select the one producing the maximum value. They are more complicated to design than the two-class versions but do not leave undefined regions. Note that the distinction between simple and piecewise discriminants is not usually made for parametric classifiers. This is because it is much easier to design the piecewise versions in the parametric case, so they are nearly always piecewise.

Nonparametric quadratic classifiers (and indeed cubic and higher order classifiers) are rarely used as their discriminant functions have a large number of terms and consequently require extremely large training sets and computation times [DH73]. We shall review only the linear classifiers.

A wide range of techniques have been used to design linear discriminants. The statistical techniques overlap with neural network training techniques. This is because the conventional model of a single neuron (with a threshold activation function (see section 6.4.1)) performs identically to the linear discriminant.

The linear discriminant function of the input vector for the two-class case is usually represented as follows:

$$d(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

where $\mathbf{w}$ is the *weight vector* and w_0 is the *threshold weight* (sometimes called the *bias* in neural network terminology). The decision function assigns $\mathbf{x}$ to class C_1 if $d(\mathbf{x}) > 0$ and to class C_2 if $d(\mathbf{x}) < 0$. The case where $d(\mathbf{x}) = 0$ represents the decision surface; the result of the decision function in this case is implementation dependent. As stated earlier, the decision surface is a hyperplane in the input space. The weight vector is normal to the hyperplane and therefore determines the orientation of the decision surface. The threshold weight determines the position of the decision surface (its distance to the origin is $w_0/|\mathbf{x}|$).

From a neural network perspective, the weights correspond to the weights on a perceptron's inputs. The decision function corresponds to the output function of the perceptron. Most methods of linking perceptrons together (to form a neural network) use layers of these units, where one layer's outputs become inputs to the next layer's units. A layer of perceptrons, all taking the same inputs, corresponds to the use of multiple discriminant functions in a piecewise classifier and results in similar performance. The use of multiple layers in the neural approach adds additional discriminatory ability which has generally distinguished the statistical and neural approaches in recent years.

The process of designing a linear discriminant or training a perceptron involves determining the weight vector and threshold weight. Strategies for this depend on whether the training set is linearly separable or not. A data set is said to be *linearly separable* if the classes can be correctly separated in the sample space by a single linear division.

When the two classes are linearly separable the general strategy is to divide them by solving a set of linear inequalities [KD]. These inequalities correspond to the decision functions of each training sample. The functions are usually simplified by including w_0 as the first element in the weight vector and making the corresponding element in the input vector '1' in each case so that $w_0 \times 1$ is always added. It is also convenient to reverse the sign of the training vectors in class C_2 so that the decision functions in all cases during training give the correct classification if $d(\mathbf{x}) > 0$. The task is then to solve the set of linear inequalities:

$$\mathbf{w}^t \mathbf{x} > 0 \quad \forall \mathbf{x} \in \textit{training set}$$

When there is a wide separation between classes many solution vectors will exist. It is desirable to choose one in the middle of the range and not one close to a class border. One strategy for achieving this is to define a positive *margin*, *b*, and keep the decision surface away from the class borders by requiring that:

$$\mathbf{w}^t \mathbf{x} \geq b > 0 \quad \forall \mathbf{x} \in \textit{training set}$$

The standard approach to solving these inequalities is to define a criterion function which is minimized if **w** is a solution vector [DH73]. The problem is thereby transformed to a search for the minimum value of the criterion function which can be found by gradient descent.

Several different methods exist for solving the inequalities, using different criterion functions and different descent procedures. A common and effective criterion function is the *perceptron criterion function* which sums the error of the discriminant functions (the amount by which they are below zero for mis-classified samples). Methods for gradient descent in the linearly separable case include the *fixed-increment rule* (Rosenblatt [Ros62]), *variable-increment rule* and *relaxation* (Agmon [Agm54], Motzkin and Schoenberg [MS54]). Duda and Hart [DH73] give clear descriptions of these methods and proof that they converge to a solution.

These methods are called *error-correction learning* procedures as they only alter the weights if a classification error occurs. The methods work when the classes are linearly separable but in practice this is rarely the case due to the large number of training samples. In non-linearly separable cases the above methods may endlessly fluctuate the weights as there will always be errors to correct. It is difficult to determine which of the fluctuating weight values to choose.

A variety of other techniques are used to determine the weights when the classes are known to be nonlinearly separable. These methods generally make a compromise between the ability to produce a linearly separating division in the linearly separable case and the ability to produce an acceptable division in the nonlinearly separable case. The methods are similar to those for linearly separable solutions but rather than solving inequalities where the discriminant functions are greater than zero, they usually try to solve linear equalities where the discriminant functions equal a constant margin value. The margin value may be different for each training sample. If the training set is not linearly separable then the equalities cannot be solved so the methods aim instead to find a weight vector that minimizes the error over the whole training set.

The most common form of this aim is to minimize the *sum-of-squared-error criterion function*, producing the *minimum-squared-error* (MSE) solution. Methods which produce this solution are called MSE procedures. The disadvantage of the MSE solution is that although it can be found in both the linearly separable and nonlinearly separable cases, it is not necessarily a linearly separating solution in the linearly separable case.

The *pseudoinverse* method (Ho and Kashyap [HK65]) is a non-iterative MSE procedure which places all the training samples in a matrix and computes the solution by solving a matrix equation. This involves use of the Moore-Penrose pseudo matrix inverse [Pen55] [Pen56]. Most other methods are iterative procedures using gradient descent. The *Widrow-Hoff rule* (or *Least Mean Square rule*) [WH60] is an iterative MSE procedure for minimizing the sum-of-squared-error criterion function.

Stochastic approximation is an area of statistical theory concerned with random sequences. A full description of stochastic approximation theory can be found in Wasan [Was69]. The theory can be applied to nonparametric classification to create an alternative iterative MSE procedure which takes samples from the training set at random according to a probabilistic law. A class C_i is chosen according to the probability $P(C_i)$ and then a sample vector $\mathbf{x}$ is chosen according to $P(\mathbf{x} | C_i)$. The criterion function used is the expectation of the squared error between the actual class of $\mathbf{x}$ (represented by +1 if it is class C_1 and -1 if it is class C_2) and the value of the discriminant function $d(\mathbf{x})$. This form of function is called a *regression function* and the iterative techniques used to solve it are called *stochastic approximation procedures*. Stochastic approximation procedures have been applied to pattern classification by several researchers, e.g., Fu [Fu68], and Yau and Schumpert [YS68].

The *Ho-Kashyap algorithm* [HK65] [HK66] is an iterative modification of MSE procedures which allows the margin values to vary. This means that in the linearly separable case the minimum value of the sum-of-squared-error criterion function is zero, and the weight vector corresponding to this zero value is a linearly separating one. This method therefore has an advantage over the previous methods as it can guarantee a linearly separating solution if one is possible. *Linear programming* solutions apply classical linear programming techniques to the problem (see Smith [Smi68]). It is also possible to apply the techniques for linearly separable training sets to nonlinearly separable cases, e.g., the fixed-increment rule (Minksy and Papert [MP69b], Block and Levin [BL70]).

Duda and Hart [DH73] give a more detailed description of all these methods and also their extension to piecewise forms. They conclude that none of them is universally superior to any of the others and which is best depends on the specific requirements of the classification problem. Consequently, all these methods are commonly used.

6.2.3. Unsupervised Classification

Where the classes of the training set samples are unknown, unsupervised classification techniques are required. Unsupervised classifiers are designed to reflect the grouping of samples, treating each group as a class. Since the classes of the samples are unknown, no meaningful class labels can be assigned (they will simply be "class 1," "class 2," etc.). It is up to the user to assign meaning to the classifier output.

There are two main techniques for designing unsupervised statistical classifiers: *unsupervised learning* and *clustering*. Unsupervised classification is not commonly applied to character recognition as it is usual for character training samples to be labelled when they are collected. Supervised techniques are therefore much more appropriate. The review of unsupervised classifiers shall therefore be kept very brief. More detailed descriptions can be found in Duda and Hart [DH73], Jain [Jai86] and Schalkoff [Sch92].

Unsupervised learning takes the same approach as supervised parametric learning (see section 6.2.1). The functional forms of the underlying class conditional density functions are assumed and their parameters are estimated based on the sample set. A popular estimation method is Bayesian estimation which, like supervised Bayesian estimation (see section 6.2.1), assumes the parameter vectors are random variables with a known *a priori* distribution. The sample set is used to determine the *a posteriori* density function. A sharp peak in this function indicates the most probable value for the parameter vector, and this value is selected for use in the classifier.

Although unsupervised learning methods are very similar to those for supervised learning, the lack of class labels on the samples complicates the parameter learning and makes these methods computationally expensive. This has led to the alternative strategy of clustering, which aims to be computationally simpler [DH73].

Clustering techniques attempt to identify classes based on the grouping of their feature vectors, on the assumption that characters with similar features will belong to the same class. The unlabelled samples are grouped into sets (called *clusters*) by one of two clustering techniques: *hierarchical* (or *iterative*) or *partitional*.

Hierarchical clustering produces a hierarchical sequence of partitions of the samples. Samples in the same partition at one level will be in the same partition in all the subsequent levels; thus a tree-like structure of clusters is formed (called a *dendrogram*). At the root of the tree is a single cluster containing all the samples, and at the top of the tree there are as many clusters as there are samples. The way in which the clusters divide up in the middle of the tree is determined by the degree of similarity of the samples in those clusters. There are two strategies for this type of clustering: *agglomerative* and *divisive*. Agglomerative strategies (also called *clumping*) start at the top of the tree and build the rest of the tree by successively merging similar samples together to form larger clusters. Divisive (or *splitting*) strategies start at the bottom of the tree and successively divide clusters into smaller ones.

Partitional clustering performs only one partioning of the sample set. The most common form of this partitioning assigns samples to clusters according to some criteria of how similar the samples are to other samples in the same cluster, and how different they are to samples in different clusters.

The criteria used in determining the splitting or clumping of samples in hierarchical methods and in determining the partitioning of samples in partitional methods are similar. The most common criteria is that the total sum of the squared distances from members of a cluster to its mean vector is minimized. Another criterion used in clustering is the ratio of the sum of the intracluster squared distances (squared distances between members of a cluster and its mean vector) to the sum of the intercluster squared distances (squared distances between each cluster's mean vector and the mean of all the other cluster means) [KD]. Other clustering criteria are the related minimum variance criteria and scattering criteria (trace, determinant and invariant criteria) (see [DH73]).

Other forms of partitional clustering are density estimators (which attempt to find cluster centroids by finding their modal values), graph theoretic methods (which link similar samples in a graph and produce clusters in the form of minimal spanning trees), and the k-means algorithm. In the k-means algorithm [BH67] (also called the c-means or Isodata algorithm) the cluster means are initially assigned at random. The samples are classified according to these means (usually by nearest-neighbour classification) and the means are then recalculated from the classification results. This procedure is repeated until a self-consistent set of cluster means is found.

The difficulty with clustering techniques is in determining the validity of the clusters. Clustering algorithms will always produce clusters even when they are not present in the data. A few quantitative measures are possible to evaluate the degree of natural clustering in the data and the validity of the chosen clusters, usually by making assumptions about the probability distributions of the classes and testing how well the clusters fit them. Ripley [Rip81] and Jain [Jai86] give good overviews of the measures used. Valid clusters can also be evaluated in terms of how "good" they are using measures of "compactness" and "isolation" [Jai86].

In addition to its use in unsupervised classification, clustering has also been used to reduce the dimensionality of data and to reduce the size of training sets (by representing compact clusters by their mean vector) [DH73].

6.2.4. Fuzzy Decision-Making Classification

Where no *a priori* knowledge is available it can be advantageous to use fuzzy decision-making classifiers for unsupervised classification. These were presented by Bezdek [Bez81] and Kandel [Kan82] as alternatives to the probabilistic statistical classifiers. The classifiers adopt the fuzzy set concept of Zadeh [Zad65] which allows an input pattern to belong to all the possible classes with varying degrees of membership. This is very useful when it is desired to output a candidate set of classes rather than a winner-take-all classification.

Bezdek [Bez81] proposed a fuzzy version of the *k*-means algorithm (see section 6.2.3) to produce classes encompassing spherical clusters of the training samples. The algorithm was later developed to produce line and surface clusters (Bezdek *et al.* [BCG81]) and clusters with variable shape (Gath and Geva [GG89]). Man and Gath [MG94] presented a variant called the fuzzy *k*-rings algorithm which forms ring-shaped clusters and can characterize both ring-shaped and compact spheroid classes.

6.3. Structural Classification

Structural pattern recognition focuses on the structural relationships within entities rather than the features of those entities. This approach assumes that the relationships can be quantified in some way. For many applications this is possible and provides a useful means of classifying patterns using higher level representations. In structural character recognition, the characters are broken down into easily extracted primitives, such as straight lines and curves. Figure 6.1 gives a simple example of horizontal and vertical line primitives. The primitives are presented to the structural classifier in an interconnected form (usually a string, graph or tree) which represents the relationships between them (relative positions and joins).

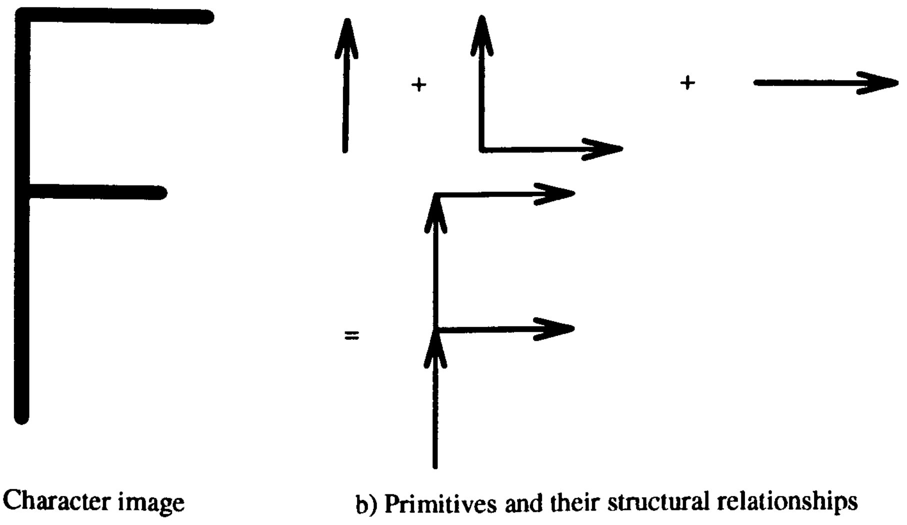


Figure 6.1 Example of primitives extracted from a character and their structural relationships.

The most simple form of structural representation is a string, which can be matched against a library using basic *string matching* techniques. Much more powerful structural representations can be achieved with *formal grammars* and *relational descriptions*. Classification by matching formal grammars is done by *parsing*; the equivalent technique for relational descriptions is called *relational graph matching*.

Hybrid techniques, sometimes called *relational models*, combine the relational representation of primitives with features, or attributes, of those primitives, to aid in disambiguation when a structure matches more than one class. The hybrid techniques are generally called *attributed* versions of the standard techniques.

The following sections describe each of these techniques as they relate to the work presented in this thesis. First though, we shall consider the most elementary form of structural classification, *template matching*.

6.3.1. Template Matching

Template matching, also known as *prototype matching* or *correlation*, was one of the first types of classification applied to character recognition and is still in common usage today. It involves storing a set of character templates, each representing a particular character class. The target character is then compared directly to each of the stored templates and the closest match is selected as the output character class. The templates can take a variety of forms. Pure bitmaps have often been used, but some form of reduction or simplification of the pixel data is useful. Typically the character is encoded into a vector, signal, string or set format [Sch92].

Template matching is considered here as a structural classification technique although some statistical classifiers reduce to a template matching procedure for certain probability density functions and certain simplified assumptions about the parameters (see Duda and Hart [DH73, p.26]). In the statistical case the templates are likely to be feature vectors, rather than structural encodings of the characters, but the operation of the template matcher remains the same.

The first problem in template matching is usually to determine the correct relative position in which to compare the template with the target. (This is not always necessary, particularly not in feature matching since features are often position invariant.) The template is shifted over the target to all possible positions. In each position a matching metric, called a *correlation function*, is computed and the best correlation is recorded for each class. The highest scoring class over all the positions is selected as the output. Commonly used correlation functions include Hamming distance, Levenshtein distance, Euclidean distance, sum-of-squared-error and weighted inner product. Some of these are specific to the format of the template.

A variant of the standard correlation approach, flexible template matching, was implemented by Lanitis and Taylor for character recognition [LT93]. Each character template had accompanying parameters (e.g., horizontal and vertical scaling parameters) which were varied, within certain constraints, to "fit" the target, allowing a wider range of slants and sizes to be contained in the template set.

Template matching is unlikely to provide useful solutions in character classification. There is no training stage so all the computation is performed when the system is in use. This makes matching very slow, even against fairly small sets. It would be impractical for matching against the large sets necessary to correctly represent all the varieties of character styles, even assuming such a set could be constructed. Template matching techniques are sensitive to the noise and distortions that occur often in real applications [GS90]. High levels of recognition accuracy have not been achieved on real data.

6.3.2. String Matching

String matching is a technique that has several applications in character recognition, not just in structural classification. One common use is in contextual processing, to match words against a dictionary. This application will be described later in section 6.6.1.

From a structural classification perspective, string matching is a specific form of template matching which can be applied to string representations of primitives. A library or dictionary of template structure strings is used to represent the structures of each class. An input string is classified by matching it against the dictionary.

This is an obvious, though impractical, approach to formal grammar matching. The string matching approach is to form a library or dictionary of all the possible strings in a grammar's language and then match the input against the dictionary. Unfortunately the languages are often very large, or even infinite, and matching is too slow for real use. A faster, and much more practical solution to matching formal grammars is parsing which will be described in the next section.

String matching suffers from the same problems as other forms of template matching. It is slow and requires very large template sets to adequately represent handwritten character classes. However, it remains an interesting approach to problems where the dictionary is small. The hybrid method of attributed string matching, which combines the structural relations of primitives with feature measurements from those primitives, is also worthy of consideration here.

The most basic form of string matching is *exact matching*, where two strings are compared, element by element, and a match is indicated if and only if all the corresponding elements are equal. This is a very limited form and is rarely used in character recognition as errors and variability are expected in both the stored strings and the input. Some measure of string dissimilarity is necessary so that input strings which are 'close' to the template strings can be recognized. This is a common extension to the basic technique, called *approximate string matching* or *error correcting parsing*.

The matching metric in approximate string matching is usually a form of *edit distance*. The edit distance between two strings is based on the operations (usually insertion, deletion and substitution of an element) required to transform one string into the other. A basic edit distance is the Levenshtein distance [Lev66] which is simply the minimum number of edit functions required for the transformation.

More complicated edit distances are used in *attributed string matching* where primitives in the string have associated features. Attributed strings have been used by several researchers for matching polygonal approximations, e.g., Tsai and Yu [TY85], Fan [Fan87] and Maes [Mae91]. Polygonal approximation (see section 2.3.2) is a common encoding of shape that can be represented in string form. The edit distances used for matching attributed strings are essentially weighted Levenshtein distances. The edit operations are weighted by a cost function on the primitive's features which provides a better measure of their dissimilarity.

The cost of edit operations depends on the encoding of the character. For example, Maes [Mae91] used feature strings of (length, angle from previous line) pairs with cost functions based on the difference between the angles and a weighted difference between line lengths for substitution operations. A simple modulus of the angle or weighted modulus of the length was used for insertions and deletions. Tsai and Yu [TY85] represented polygons as strings of (length, angle from a starting line on the polygon) and defined cost functions as a sum of angle change and a measure of proportional change in length.

It is common in polygonal approximation for two, roughly identical, line segments to be represented by differing numbers of boundary primitives. To overcome this problem, Tsai and Yu [TS85] added an extra edit function, called "merge," which combines any number of consecutive string elements into a single element (i.e., a single boundary primitive) for matching against a template.

Matching records the edit distances between the input and all the strings in the template set. The template string with minimum edit distance is selected as the output class. This can be seen as a variant of nearest neighbour classification which operates on strings rather than feature vectors [Sch92].

The possible ways of matching two strings in terms of the edit operations required can be visualized as a graph. Each vertex represents a state of the match and has three arcs leading to other states, one arc for each of the possible edit operations. The problem of finding the closest match becomes one of finding the shortest path from the start state to the finish state.

Maes [Mae90] [Mae91] extended conventional linear string matching to cyclic strings. Cyclically shifting the strings provides rotational invariance of recognition and eliminates the problem in linear matching of determining an analogous starting point on the polygonal approximations of different characters. By eliminating parts of the match graph that need not be searched, Maes reduced the complexity of the cyclic matching algorithm from $O(n^3)$ to $O(n^2 \log n)$ for strings of length n. The time could be reduced further by assuming the correct orientation is approximately known and only testing a few orientations close to the vertical.

Another approach to string matching is matching with k differences. This is faster but provides less information to disambiguate different closely matching templates than edit distance matching.

String matching, in general, is very slow. It suffers from the same problem as nearest neighbour classification — matching against large sets makes the matching too slow for practical use. Large sets are inevitable in character recognition as the templates must cater for the wide variation in character styles. The applications of attributed string matching to polygonal approximations have only proved effective for recognizing simple shapes where the template sets are small. The speed problem, coupled with the other deficiencies of polygonal approximations (see section 2.3.2), make this an unlikely solution to isolated character classification. Marzal and Vidal [MV93] used a string matching technique on a small set of handwritten digit chain-codes and achieved reasonable results (over 90%) but again this is unlikely to be as accurate or practical on a large data set. The techniques used are interesting, however, as they are also used in certain contextual processing approaches. This is discussed further in section 6.6.1.

6.3.3. Formal Grammars and Parsing

Formal grammars and parsing make up a type of structural pattern recognition sometimes referred to as syntactic, grammatical or linguistic pattern recognition due to its basis in formal language theory (see Hopcroft and Ullman [HU79]). Formal grammars originated in the work of Chomsky [Cho57] and have been the subject of much interest since then. Their application in structural pattern recognition is described in detail by Gonzalez and Thomason [GT78], Fu [Fu82], Miclet [Mic86] and Schalkoff [Sch92].

Formal grammars describe the allowable patterns, called *sentences*, in a *language*. Although they are often used to model spoken and written languages, such as English, the terms language and sentence, as they apply to grammars, have a much wider scope. In the case of character recognition, the language means the whole range of structures belonging to a particular class or subclass of character. A sentence is an instance of a language's structure.

Grammars consist of symbols and productions (also called production rules and rewriting rules). There are three types of symbols: a starting (or root) symbol from which the generation of a sentence starts; terminal symbols (or primitives) which make up the final sentence; and non-terminal symbols (or variables) which are always rewritten during the generation so that only terminal symbols remain in the final generated sentence. The terminal symbols correspond to structural primitives; the non-terminals correspond to more complex subpatterns which are successively constructed from primitives. The productions are substitution rules which are applied to the starting symbol (and subsequently to any non-terminals) in a hierarchical fashion to generate sentences.

The most common types of grammar are *string grammars* where primitives and their relations are written in a string. Chomsky [Cho57] proposed four types of string grammar, each with different constraints on the allowable productions. The four types are, in order of increasing constraints, *unrestricted* (or *free*), *context-sensitive*, *context-free* and *finite-state* (or *regular*). Unrestricted grammars have no constraints. Context-sensitive grammars may substitute symbol strings in the context of the symbol strings on either side. Context-free grammars are constrained to only substitute for single non-terminal symbols (the non-terminals are replaced independently of their context). Finite-state grammars may only have one non-terminal symbol on either side of a production rule. Character recognition applications usually use context-free grammars, which have reasonable descriptive capability and can be efficiently parsed. Occasionally the more powerful but more complex context-sensitive grammars are used, e.g., Stringa [Str90].

As well as generating the sentences of a language, grammars may also be used to determine the syntactic structure of a given sentence and whether it belongs to a language. This is generally how grammars are used in classification. An input structure is classified by determining which class's language it belongs to. This analytic use of a grammar is achieved by parsing (also called *syntactic analysis*).

Parsing can be either top-down or bottom-up. Top-down parsing starts with the start symbol and performs productions from the grammar to try to generate the input sentence. Bottom-up parsing starts with the input pattern and applies the production rules backwards to try to reduce the sentence to the start symbol. A successful parsing of a grammar indicates that the sentence belongs to the associated class. The productions used in the correct parsing of a sentence indicate its syntax and provide a descriptive understanding of the input.

Among the main techniques for parsing string grammars are the Cocke-Younger-Kasami algorithm (see Schalkoff [Sch92]), Earley's algorithm (see Aho and Ullman [AU72]), transition networks and augmented transition networks (Woods [Woo70]]).

Representing the relationships of primitives by concatenating them in a string is limited in that it only describes one dimension of positional relation. Shaw [Sha70] allowed greater descriptive ability in his Picture Description Language using mathematical operators to indicate the direction of picture primitives (using the negation operator to indicate reversal of the primitive's head and tail) and the various types of joins (tail to head, tail to tail, head to head and tail to tail). This is still a restricted representation for two-dimensional character images as it cannot represent some topological relations involving three or more primitives. More complex, higher order grammars are required.

The most common higher-dimensional grammars used for two-dimensional character recognition are *tree grammars* (see Gonzalez and Thomason [GT78], Fu [Fu86] and Schalkoff [Sch92]). These use trees to represent structure. A tree is a directed, acyclic graph whose nodes represent the primitives and whose arcs represent their relationships. Multiple arcs from a node may correspond to relationships (joins) at different points on the primitive. The simplest type of productions for tree grammars start with a whole tree structure whose nodes contain non-terminal symbols. The non-terminals may be replaced as for string grammars but no new arcs or nodes may be added. This eventually produces a tree with the same structure but containing only terminal symbols. More powerful rules are *expansive production forms* which allow non-terminals to be replaced by new arcs and nodes so that the trees can be grown from a single root node. Parsing is performed using tree automata.

The other main types of higher order grammars are *plex grammars* (see Gonzales and Thomason [GT78] and Fu [Fu82]), *web grammars* (Pfaltz and Rosenfeld [PR69]), *graph grammars* (Pavlidis [Pav72]), multidimensional *array grammars* (Rosenfeld [Ros73], Wang [Wan80]) and *mosaic grammars* (Ota [Ota75]).

The practical difficulty with syntactic character recognition is in determining grammars to accurately represent the huge variety of style in character classes. Grammars are often defined by a human designer but there are techniques, called *grammatical inference* procedures, which automatically construct a grammar to represent a training set. Grammatical (or structural) inference is the equivalent of learning in a statistical or neural classifier. Among the proposed methods for string grammars are grammatical inference using lattice structures, structural information sequences and inductive inference (see Fu [Fu82], and Fu and Booth [FB86]). Learning of tree grammars has been attempted by Brayer and Fu [BF77] and Levine [Lev81]. Winston [Win70] proposed a grammatical inference method for web grammars.

The problems faced by both human designers and grammatical inference procedures are caused by three main factors. Firstly, a class language may be generated by many different grammars. There is therefore a choice to be made about which of the possible grammars is best. Secondly, grammars will often generate sentences that do not belong to the intended language, so the next problem is to restrict these unwanted sentences. The third problem with the grammars we have considered so far is that they produce a simple yes/no result — a sentence (character) either fits the grammar (class) or it doesn't. For this reason, it is usually desired that class languages are disjoint so that only one will be matched for any given sentence.

Grammatical inference methods (and possibly human methods too) address the first problem by restricting the grammar to a standard type, e.g, context-free, or normal form. Other constraints are also used such as requiring the grammar to be of minimal complexity [Sch92]. The problem of unwanted sentences is addressed, in part, by including negative exemplars in the training set, i.e., sentences that do not belong to the language. The inference methods then produce grammars that match the positive exemplars but not the negative ones.

The third problem is the most difficult to overcome for character recognition and is the main weakness of the syntactic approach. The difficulty is that handwritten character classes are not disjoint, particularly when represented structurally by discretized stroke primitives. The simpler the set of primitives the greater the amount of overlap. This means that a non-overlapping division by grammars is a matter of compromise, for which no solid theory exists. Statistical classifiers provide a much more practical and theoretically sound solution to the problem. Another weakness of most syntactic pattern recognition techniques is that they consider character structures to be pure forms, obeying fairly rigid rules. This is not the case for real world applications where distorted and noisy images are commonplace. In addition, syntactic recognition does not consider the possibility of errors in the primitive extraction process. Extraction errors often occur, especially when processing noisy images. Formal grammar approaches have therefore had limited success in character recognition as they are not powerful enough to cope with these problems in real data.

One development of formal grammars that addresses these practical problems, however, is *stochastic grammar* (Booth and Thompson [BT73]). Stochastic grammars allow class languages to overlap but incorporate probabilities into their production rules and choice of starting symbol. Sentences generated by stochastic grammars have an associated probability, derived from the probabilities of the productions used in their generation, which serves as a certainty measure of how well the sentence belongs to the class. This provides a useful measure to aid in the disambiguation of sentences which match more than one class's grammar. Stochastic grammars have also been extended to trees (Fu [Fu80], Thomason [Tho86]).

Another hybrid extension is *attributed grammar* (Knuth [Knu68], Sebesta [Seb89]) where symbols have associated features or attributes. These attributes also influence the production rules. This allows more powerful descriptions of characters, such as are achieved by conventional feature-based approaches. Ali and Pavlidis [AP77] applied attributed grammars to the classification of 840 handwritten digits, encoded by polygonal approximations (see section 2.3.2), and achieved a correct recognition rate of 92.98%. Their method is limited, however, in that the polygonal approximation can only represent one component of a character. On a larger data set, including broken characters, a high (about 30%) error rate occurred.

6.3.4. Relational Descriptions and Relational Graph Matching

In cases where the number of training samples is too small to accurately infer class grammars, or where individual patterns can be considered to be representative templates (or prototypes) of their classes, relational descriptions are often a better alternative to formal grammars [Sch92]. Relational descriptions are classified by template matching rather than parsing. Forms of template matching for classifying simple structural representations, such as strings, have already been discussed in sections 6.3.1 and 6.3.2. As mentioned in section 6.3.3, higher-dimensional representations are useful for describing the full range of relationships in two-dimensional character images. This section briefly reviews the matching methods for higher-dimensional relational descriptions. The main types of higher order descriptors are trees and graphs, but since trees are a subclass of graphs we shall only consider graphs in this section.

Relational description using graphs is an extension of higher-dimensional grammars [Sch92]. Structural patterns are represented by a specific type of semantic net called a *relational graph*. A semantic net is a labelled, directed graph which describes relationships and properties of entities. In the relational graph, nodes represent subpatterns or structural primitives, and the directed arcs of the graph represent the structural relationships between nodes.

A template set is used to represent character classes. Relational graph matching determines which class's template best matches the input graph. The simplest form of matching is exact matching where graphs must be *isomorphic* for a match to occur. An isomorphic map preserves the structure of the graph. In other words all nodes and arcs of the graphs are identical, except possibly for their labels.

Isomorphism is usually determined using the *adjacency matrices* of the two graphs. An adjacency matrix is an encoding of the graph with one row and column for each node. The matrix contains '0's or '1's to indicate whether or not the row and column nodes are related. The corresponding elements of each matrix are compared; if they are all identical then the graphs are isomorphic. For one matrix this can be achieved in $O(n^2)$, where n is the number of nodes, but each ordering of the nodes must be tested, making the complexity of the whole process $O(n^2.n!)$. This degree of computational complexity is the main problem with the relational graph approach. One attempt to reduce the complexity involves determining if two graphs are *not isomorphic*, by looking for invariant properties of isomorphic graphs (number of nodes, number of incoming and outgoing arcs from nodes, closed paths) that are not met by the target graphs.

In practice, character structures do not always match a template graph so precisely. Noise and distortion cause local and structural deformations of the input graphs. Some measure of similarity must be used to allow inexact matching of noisy structures.

Sanfeliu and Fu [SF83] review the two main methods for introducing similarity to graph matching. The first is to extract features from the graphs and match them using statistical classification. This relies on the extraction of good features, but the types of features used for this purpose are generally simple counts of particular substructures, e.g., number of nodes, number of nodes with n arcs, number of triangles. The simplicity of the countable features limits the power of the structural representation so these methods are not greatly different from conventional, feature-based, statistical classification techniques. The second method uses the graphical equivalent of Levenshtein distance, i.e., the minimum number of edit operations to transform one graph into the other, where operations are, for example, node insertion, deletion, splitting and merging, and arc insertion and deletion. It has been shown that this method has polynomial time complexity for trees [Lu79], but for graphs is, in the worst case, exponential [SF83].

The weakness of relational graph matching is the same as for other types of structural character classification; structure alone is frequently insufficient to discriminate characters. As with the other forms of structural representation there are hybrid versions of relational graphs which include features of the nodes (primitives) to aid in disambiguation of classes with similar structures. These are called *attributed graphs.*

Similarity measures between graphs are naturally extended to attributed graphs by weighting the cost of edit operations according to the attributes of the transformed nodes. This is similar to the weighting of operations for attributed string matching (see section 6.3.2). However, attributed graph matching is more complicated than attributed string matching because in addition to comparing nodes' attributes it is also necessary to determine the best pairings of nodes for comparison.

Good candidates for pairing may be found using the *maximal clique* method. A clique of a graph is a totally connected subgraph. Totally connected means that there is a direct path between any two nodes in the subgraph. A maximal clique is one which is not contained in any other clique. The method starts by producing a *match graph* between the input and template graphs. The nodes of the match graph represent possible pairings of input graph nodes with template graph nodes. Nodes in the match graph are linked with an arc if the pairings they represent are compatible. Maximal cliques in the match graph indicate good candidate pairings for matching the input to the template. Again, this method is very computationally expensive. Davies [Dav90] reports that maximal clique is NP-complete.

An alternative approach is *probabilistic relaxation*, also called *continuous relaxation* (Rosenfeld *et al.* [RHZ76]). Relaxation is an iterative procedure where matches between input and template nodes are assigned probabilities of their correctness. (A discrete version of relaxation is also possible where matches are assigned a simple yes/no value rather than a probability.) The probabilities are initially based on similarity measures between the attributes of the nodes. They are then iteratively modified according to the context of their adjoining nodes. Suppose we have a potential match between a node, *i*, in the input graph and a node *t* in the template graph. If a node linked to *i* in the input graph has a high match probability with the corresponding node linked to *t* in the template graph, then the match probability between *i* and *t* is increased. Similarly, if the match probability between corresponding linked nodes is low, the match probability between *i* and *t* is reduced. After a number of iterations the high probability matches should indicate the best pairings. Many relaxation techniques have been proposed. Kittler and Illingworth [KI85] give a comprehensive survey of techniques and applications.

Sanfeliu and Fu [SF83] applied attributed graph matching to a small set of 25 handwritten lower case 'b's, 'd's, 'h's and 'k's with good results. The computational complexity would limit its application to large template sets however, and the small scale of the test prohibits meaningful comparison with other systems.

Yamamoto and Rosenfeld [YR82] used probabilistic relaxation to match attributed graphs for handprinted Kanji characters and Lam and Suen [LS88a] developed the method for application to handwritten digit recognition. They used a set of heuristic rules to simply classify characters consisting of only one or two primitives. This reduced the number of characters required to be matched by relaxation and hence improved the speed of the overall recognition system. A set of 5000 samples, taken from real data, was used to train the simple classifier and aid in construction of the templates for graph matching. The combined classifiers achieved high recognition rates (between 93 and 96% depending on the rejection threshold used) on 2000 unseen test samples.

Lu *et al.* [LRS91] used *branch attributed graphs* to represent the structure of branches of Chinese characters. They ordered the vertices of the branch attributed graphs and then organized them hierarchically to facilitate a search procedure. This organization helped reduce the complexity of matching. They applied this relational graph method (which included attributes for both nodes and arcs) to handwritten Chinese character recognition and achieved a 90.45% correct recognition rate on a sizeable test using good quality data.

6.4. Neural Classification

Neural classification is an emerging field of classification research which has become extremely popular in recent years. Artificial neural networks, also known as connectionist models, neuromorphic systems and parallel distributed processing models, are much more powerful classifiers than the conventional statistical and structural approaches. It is still not clear, however, that neural classification is an entirely separate paradigm as many neural methods are closely related to statistical and structural ones.

Since 1943, when McCulloch and Pitts first proposed their model of a neuron [MP43], many different neural classifiers have been proposed. Neural classification techniques were applied to pattern recognition problems very early in their development. Most, if not all, of the methods described here have been applied to character classification at some point.

Artificial neural networks model the function of real neural networks in the human brain. They comprise units, called *neurons*, connected together in a network topology. The function of the neurons, and in some cases the structure of the networks, can be trained to learn sets of patterns. Individual artificial neurons model the function of biological neurons which is reasonably well understood in the field of neuroscience. The network structures and the training methods are also generally intended to model their equivalents in the brain, though these are less well understood.

Artificial neural networks (hereafter referred to simply as neural networks) take a wide variety of forms but certain generic structures have dominated the field. The following sections review the different models of neuron function, the types of network structures and the training algorithms used. Finally some specific neural classifiers which have been used for character recognition are surveyed.

6.4.1. Types of Neuron

The basic model of a neuron is as shown in figure 6.2. The neuron has a number of inputs $(1,...,n)$ and a single output. The values of the inputs $(x_1,...,x_n)$ and output (o) are called *activations*. Each input has an associated weighting factor $(w_1,...,w_n)$, called simply a *weight*. The weighted sum of the input activations (*net*) gives the equivalent of the *soma potential* of a biological neuron. This is passed through an *activation function* $(f(net))$ to obtain the output activation.

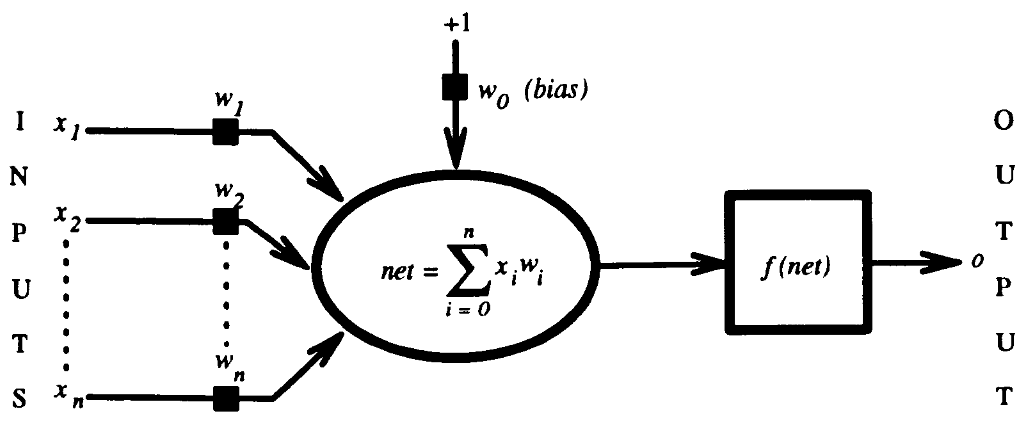


Figure 6.2 Basic model of a neuron.

The neurons that have been used in pattern classification vary in several ways. Firstly the input and output activations may be either discrete or continuous. The activations are usually restricted to the range 0 to 1 (or in some cases -1 to 1). Thus the discrete case usually means the activations are binary (either 0 or 1). Discrete activations can simplify the training and operation of neural networks but are less powerful from a classification point of view as they can't be used with real-valued features.

Weight values may also be either discrete or continuous, though continuous weights are by far the most common. Early neuron models did not actually have weights; the soma potential was simply the sum of the inputs. In the neuron model in the figure, this corresponds to all the weights having the value 1. Note that in addition to the weights on the inputs, most neurons also have a threshold weight (w_0), often called a *bias* or *offset*. This is effectively an additional input whose activation is always +1. This bias is essential as it effectively determines the position of the neuron's decision surface relative to the origin of the feature space. Without it the surface would always have to pass through the origin which greatly restricts its discriminant ability.

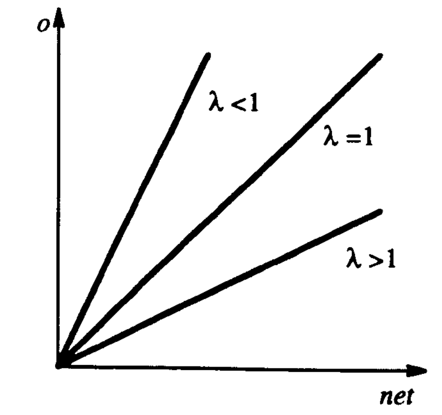
The last way in which neurons vary within the standard model is in their activation functions. The most common forms of activation functions are *linear*, *threshold* (also called *binary* or *step*), *threshold-linear* (also called *ramp*) and *general sigmoid* (also called *logistic* or *squashing*), as illustrated in figure 6.3. The type of function which can be used may be limited by the type of activation allowed (discrete or continuous).

Neurons with threshold activation functions are equivalent to linear discriminants. The other types of function are usually more useful in character classification as they produce a continuous score representing how well the input matches the output class. This can be treated as a confidence factor, or in some cases a probability, and aids in selecting the overall classification from the set of neuron outputs (usually there will be one neuron for each output class). The sigmoid function is more commonly used than the threshold-linear function as the popular back-propagation learning algorithm requires the activation function to be differentiable. Sigmoids with $\lambda = 1$ (see figure 6.3) have a simple derivative, $o(1-o)$. Also, it has been found that the average output activation of a biological neuron is a sigmoidal function of its soma potential, giving a degree of validity to this choice of function.

The main variant of the standard neuron model uses *radial basis* activation functions (Broomhead and Lowe [BL88], Moody and Darken [MD89]), also called *localized receptive fields*, *locally tuned processing units* and *potential functions*. These produce a maximum value at zero and approach zero at infinity. Rather than multiplying the inputs by the weights, these activation functions take as their input the distance between the input vector and the weight vector. The neurons therefore produce a measure of how close the input is to the learned weight vector. Radial basis function networks are discussed further in section 6.4.4.

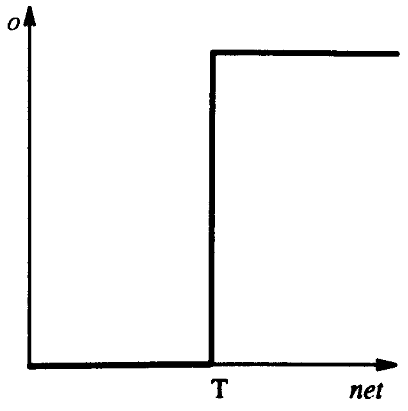
6.4.2. Types of Network

A single neuron is not a particularly powerful processing unit but when many of them are connected together (by linking one's outputs to another's inputs) they can form extremely effective systems, the most powerful of which is the human brain. Neural networks, as they are used in character recognition and other pattern recognition applications, take the form of *pattern associators* (PAs). Each PA takes an input pattern (feature vector) and produces an output pattern or class. There are essentially four forms the output may take:



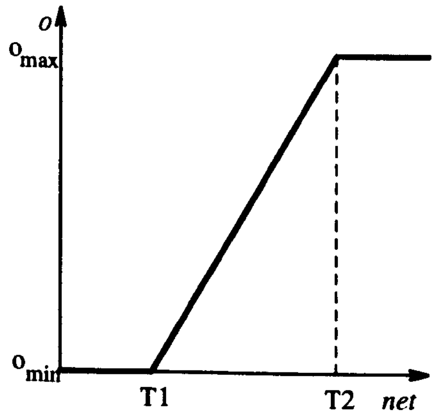
a) Linear.

$$o = \lambda \, net$$



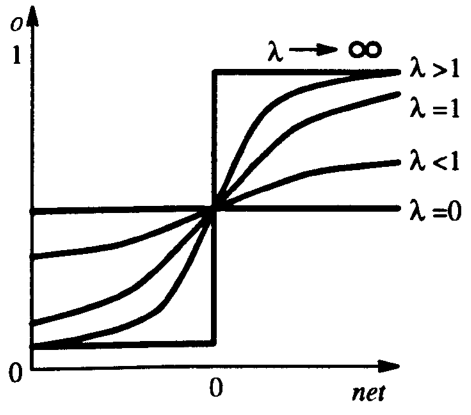
b) Threshold

$$o = \begin{cases} 1 & net >= T \\ 0 & net < T \end{cases}$$



c) Threshold-linear

$$o = \begin{cases} o_{max} & net >= T2 \\ \dfrac{o_{max} - o_{min}}{T2 - T1}(net - T1) + o_{min} & T1 <= net < T2 \\ o_{min} & net < T1 \end{cases}$$



d) Sigmoidal

$$o = \frac{1}{1 + e^{-\lambda \, net}}$$

Figure 6.3 Common forms of activation functions.

1) a single numeric output with different values corresponding to different classes;

2) multiple numeric outputs (one for each possible class) with only one producing a non-zero response for any given input pattern;

3) multiple numeric outputs which when taken together form an encoding of a class;

4) a single pattern vector.

All these are types of *content addressable memories* (CAMs). CAMs are memory architectures which are addressed by their content rather than by an index. The fourth variant is a particularly useful type of CAM called a *CAM associator.*

Pattern associators have two main mapping mechanisms. The *autocorrelator* (or *autoassociative* structure) stores a set of single patterns. These are used mainly for simple recall or correction of patterns, and completion of partial patterns. The *heterocorrelator* (or *heteroassociative* structure) stores pattern pairs, each pair consisting of an input (or stimulus) pattern and the correct response pattern. Heterocorrelators are commonly used in classification as the response patterns can be classes, which means that the heterocorrelator maps input patterns (usually feature vectors) to classes.

There are two main forms of recall. *Nearest-neighbour* recall finds the stored stimulus pattern 'closest' to the input and returns the corresponding response pattern. (This assumes a heteroassociative recall; autoassociative recall simply returns the 'closest' stored pattern.) Some associators perform an *interpolative recall* where the output is interpolated over all the stored patterns, rather than just the closest one. Note that 'closeness' may be measured in a variety of ways (though the Euclidean and Hamming distances are the norm).

Neural networks have been used to implement all these types of pattern associator. Network structures fall into two categories: *feedforward* and *feedback* (also called *recurrent*).

Feedforward structures consist of ordered layers of neurons, with the outputs of each layer only connected to inputs of neurons in the following layer. The first layer is simply a storage layer to hold the input values to the network. Neurons in the other layers perform the standard neuron functions as described in section 6.4.1. The final layer produces the network outputs. The middle layers, if any, are called *hidden layers* and the neurons they contain are termed *hidden units.* Processing in these networks is directed from the inputs through the hidden layers to the outputs and is therefore called feedforward.

Feedback networks are similar but allow the output of any neuron to be fed to the input of any other neuron. Feeding inputs back to previous neurons allows recursion (loops) in the processing sequence of the network. Neuron activations in the feedback network may fluctuate as a result of these loops, so one of the main concerns in their development is whether or not the activations converge to a stable state.

Neural networks have been formulated in many different ways but one common formulation uses matrices to store the connection weights. Generally the weight in the *i*th column and *j*th row of the matrix is the weight of the connection between the *i*th neuron's output and the *j*th neuron's input. The value zero generally indicates no connection, although some training algorithms can cause connected neurons to have a connection weight of zero. Using this formulation, computation of the soma potential is performed in a simple matrix multiplication of the input activations by the weight matrix. The matrix formulation is particularly useful in networks which use linear activation functions as the whole mapping from input to output can be performed with matrix operations. It also allows linear algebra methods to be applied to the operation and training of linear networks.

The following section describes some of the main network structures in relation to their learning algorithms. Other types of neural network are described in sections 6.4.4 and 6.4.5.

6.4.3. Types of Learning

Learning in neural networks can be divided into supervised and unsupervised in the same way as for other classifiers. Supervised learning can be further divided into *structural learning* and *temporal learning*. Structural learning simply trains the network weights so that they learn a sample set of mappings. Temporal learning teaches the network a sequence of patterns and is less commonly used in character recognition, although it could be useful for learning sequences of letters which make up valid words. This review will concentrate on structural learning techniques.

The main supervised learning procedures are *error-correction learning*, *reinforcement learning* and *stochastic learning*. The main unsupervised procedures are *Hebbian learning*, and *competitive and cooperative learning*. These are described below. The learning procedures are in some cases linked to specific network structures so several networks are also described in this section.

The most popular supervised learning techniques are types of error-correction learning. These have an interesting history.

In 1962, Rosenblatt [Ros62] proposed the use of *back-coupled error correction* to adapt the weights of a type of neural network called a "perceptron" [Ros58]. (The original use of the term referred to the whole network but it is now commonly used to refer to a single unit or neuron.) An error measure is determined by subtracting the output vector of the neural network from a target vector. This error is fed back to the network's weights which are modified according to the error vector and some probabilistic law. This is often called "perceptron learning."

Note that perceptron learning is the same as the fixed-increment rule for determining the weight vector in a nonparametric statistical classifier. As explained in section 6.2.2, the perceptron is functionally identical to a linear discriminant function, excepting the nonlinear activation functions. Consequently, many of the neural network learning rules have similarities to the nonparametric statistical learning rules. Further comparisons of neural and statistical classification can be found in Cheng and Titterington [CT84], Michie *et al.* [MST94] and Sarle [Sar94]. Neural network researchers are often criticized by statisticians for reinventing statistical models, and for training and using neural networks with no regard for the underlying assumptions made about probability distributions.

Another method which was mentioned in section 6.2.2 is the Widrow-Hoff rule. Widrow and Hoff [WH60] modified Rosenblatt's back-coupled error correction to make it minimize the mean squared error over all inputs. This method was used to train their Adaline and Madaline systems [WW88]. An Adaline is a type of two-layer perceptron network where there is only one neuron in the second layer; a Madaline has multiple neurons in the second layer. The Least Mean Square (LMS) algorithm is basically the same as perceptron learning but is suitable for continuous activations of the neurons whereas perceptron learning is for neurons with discrete binary activations. The error term is the difference between the soma potential and the target output rather than the actual output and the target output.

Multilayering of networks using sufficient hidden layers enables perceptrons to form arbitrarily complex decision boundaries. Palm [Pal79] showed that a single layer of hidden units is sufficient for a network to perform nonlinearly separable functions. However, in 1969 the field of neural networks went into a temporary decline as a result of Minsky and Papert's book [MP69b] which pointed out the perceptron's apparent inability to learn these nonlinearly separable functions. The problem, known as the *credit assignment problem*, was that at the time there was no method for computing the error of the hidden units and therefore no way to train them. Interest has grown again in recent years as ways have been found to solve the problem. A little known fact is that a mathematical solution to the credit assignment problem was published prior to Minsky and Papert's book by Amari [Ama67] but went unnoticed.

Back-propagation, also called the *generalized delta rule*, is a development of back-coupled error correction, first discovered by Werbos [Wer74]. This went unnoticed as well. It combines the least mean squared error correction of Widrow with a derivative term to keep weights within certain bounds. The derivative of the output function is used so this function must be differentiable; a sigmoid function is the usual choice. In addition to this error correction, a "weight transport" system is used where errors are propagated back to lower layers of the MLP. The hidden units of the MLP have no obvious semantics. For these neurons there can be no known target output on which to base an error measure. Instead errors are propagated back from higher layer neurons in proportion to the weight from the hidden cell to the higher layer neuron, relative to the higher layer neuron's other inputs. Back-propagation is described in more detail in the classic reference, Rumelhart and McClelland [RM86].

While back-propagation is the most popular neural network training algorithm, it is very slow. It also is prone to becoming stuck in local minima of the associated energy surface rather than finding the global minimum.

The energy surface [Den86] [HT86] is a way of understanding learning in the network. It is the surface of an energy function, or cost function, whose domain is the state space of the network, a hypercube; it represents "computational energy." The most common cost function is the squared error — the squared difference between the desired and actual outputs.

Linear error (Alstyne [Als88]) and higher powers of the error (Hanson and Burr [HB88]) have also been used. Another common energy function is an information theoretic cross-entropy measure (Ackley *et al.* [AHS85], Baum and Wilczek [BW88], Wright [Wri89]). The role of learning is to minimize the cost function. This is achieved by a gradient descent down the energy surface to a minima. The best solution is found at the global minimum so it is undesirable to become stuck in local minima when training.

An important research aim is to make back-propagation faster. Tollenaere [Tol90] proposed SuperSAB, an algorithm for adaptive acceleration of the gradient descent. Fahlman [Fah88] proposed an algorithm called *quickprop* which attempted to solve many of the learning problems of back-propagation. Second-order error derivatives are used in addition to the first-order ones to give an indication of how fast the energy surface can be descended.

Fahlman used quickprop in his cascade-correlation networks [FL90] [Fah91] [HF92] although these are very different from standard MLPs as they do not back-propagate errors to the hidden layers. Hidden layer cells have fixed input weights, determined by training a cache of cells and choosing the one with the greatest correlation between its value and the error at the output. This is similar to the method used by Gallant in his connectionist expert systems [GS87] [Gal88]. A recurrent version of cascade-correlation networks has also been developed [Fah91].

There has been much research into back-propagation for MLP networks and they have been used to achieve high accuracy results on character recognition by many researchers, e.g., Burr [Bur86], Pawlicki *et al.* [PLH88], Khotanzad and Lu [KL88], Rajavelu *et al.* [RMS89], and Sabourin and Mitiche [SM92]. However, the fact that they learn very slowly, particularly for the large networks that are required for real world problems like character recognition, still poses a problem. Speed of adaptation is an important consideration in a character recognition system, which may be expected to regularly learn new users' handwriting. Parallel algorithms are desirable for practical use of back-propagation.

There are some interesting possibilities though, such as Wright's probabilistic neural networks based on MLPs with a back-propagation-like training algorithm [Wri89]. These could enable the classifier to select from a range of high probability outputs. Learning is performed by a gradient descent to minimize an information theoretic cross-entropy measure. The activation function is chosen to form a Markov model (see section 6.2.2) of the network's layers. This simplifies the learning condition and gives a learning rule very similar to the generalized delta rule for training back-propagation networks.

Reinforcement learning (Widrow *et al.* [WGM73], Williams [Wil87]) is similar to error-correction learning but uses only one value to indicate the output layer's error rather than one error value for each neuron. The change in each weight is proportional to the difference between the output layer's scalar error value and a reinforcement threshold which is specific to each output neuron. The weight change is also scaled by the *canonical eligibility* of the weight. The canonical eligibility is a probabilistic measure of whether the computed output equals the target output. This is in contrast to error-correction learning where the difference between computed and target outputs is determined exactly.

Stochastic learning (Ackley *et al.* [AHS85]) uses random processes to change the network weights. The energy function of the network is evaluated to see if the weight change has caused a decrease in energy (i.e., an improvement in performance). The weight change is accepted if the energy decreases. If it increases, the weight change is accepted or rejected according to some probability. Although this allows acceptance of changes which give worse performance, it is useful because it enables the learning process to escape local minima in the energy surface. This is called *simulated annealing* (Kirkpatrick *et al.* [KGV83], Laarhoven and Aarts [LA87]). The acceptance probability is slowly reduced to enable the process to settle at the global energy minimum.

Hardwired learning, where the designers set the network weights themselves, is also a form of supervised learning. This is a difficult task and not commonly used although a few researchers have hardwired networks to learn language semantics which may have some usefulness in character recognition (Cottrell and Small [CS83] [CS84], Fanty [Fan86]).

Hebbian learning is a common approach, mainly used in unsupervised learning. Hebb's scheme [Heb49] increases a connection weight where there is a positive correlation between its corresponding input and output activations (in neurological terms, where there is a correlation between presynaptic and postsynaptic activity). In other words, if the input and output activations are both positive or both negative then the weight is increased. This is called *simple Hebbian learning*. The problem with this method is that if the weight is only ever increased, "synaptic strength" saturation can occur — the weights reach their maximum value.

Many variants of Hebbian learning exist which allow negative modification of the weights and correlate different values. Sejnowski [Sej77a] [Sej77b] correlated the covariance of the input and output activations, Sutton and Barto [SB81] correlated the variance of the output with the mean of the input. Klopf's *drive-reinforcement learning* [Klo86] correlates the changes in input and output. These are all discrete time methods. Continuous time methods include a decay term and calculate the derivative of the weight with respect to time, rather than a discrete weight change. Grossberg [Gro68] used a continuous time method which performs a nonlinear threshold function on the activations before correlating them. This is called the *signal Hebbian learning law* or the *passive decay associative law*. Kosko's *differential Hebbian learning* [Kos86] correlated the derivatives of the nonlinear threshold functions. Tesauro [Tes86] gives a more detailed description of the Hebbian variants.

Hebbian learning computes modifications of the weights using only the input activation, soma potential and existing weight value. It is therefore simpler to perform than back-coupled error correction which requires computation of a non-local term to determine the weight modifications.

Competitive learning (Grossberg [Gro72] [Gro76a] [Gro87], von der Malsburg [Mal73], Rumelhart and Zipser [RZ85]) is a mechanism for unsupervised training of networks such as the *additive Grossberg* [Gro68] and *shunting Grossberg* (also called *multiplicative Grossberg*) [Gro73]. Networks for this type of learning are usually either single-layer or two-layer. Neurons in the output layer of these networks have recurrent self-excitatory connections, which increase their own activations, and inhibitory connections to each of their neighbours. Excitatory connections send positive signals; inhibitory connections send negative signals. This arrangement is sometimes called *on-centre/off-surround*. On-centre refers to the self-excitatory connections and off-surround refers to the neighbour-inhibiting connections.

The usual operation of competitive learning involves an initial recall mechanism. A training sample is input to the network. In the two-layer case this input feeds forward, via weighted connections, to the output layer. The output layer then sends signals via its self-excitatory and neighbour-inhibitory connections producing a competition between the neurons. After this recursive competition, the network will settle to a state where only one output-layer neuron remains active. The training sample is then considered to belong to the class of the winner in the output layer. The weights to this winner are then modified by the *gated decay long term memory* equation

$$\dot{w}_{ij} = S(b_j)(-\alpha w_{ij} + \beta S(a_i))$$

where a_i is the activation of the *i*th input-layer neuron, b_j is the activation of the *j*th output-layer winner, w_{ij} is the weight from the *i*th input-layer neuron to the *j*th output-layer winner, and S() is a sigmoid function [Sim90]. Note that single-layer networks are usually symmetric ($w_{ij} = w_{ji}$) so both weights are modified together. In two-layer networks, only the feedforward connections to b_j are changed. This can be seen as contrast enhancement [Gro76a].

Cooperative learning is an extension of competitive learning where neighbours can be excited as well as inhibited. It was introduced by Grossberg [Gro73] [Gro82]. His shunting networks have separate excitatory (positive) and inhibitory (negative) inputs to each neuron. The learning procedure is very similar to competitive learning but the dynamics of recall are changed by the distinction between positive and negative inputs. Gain control is incorporated into the recall, which amplifies signal and nullifies noise. Grossberg's further development of competitive-cooperative networks (see section 6.4.5) also includes this gain control.

The advantages of this form of competitive (or competitive-cooperative) learning are that it is unsupervised and allows on-line training (learning while the system is in use) so *a priori* exemplars are not required. The difficulties are that storage capacity is limited and new exemplars can overwrite previously learned ones. Grossberg called this the *stability-plasticity dilemma* but he overcame it in his adaptive resonance theory networks (see section 6.4.5). A number of other forms of competitive learning have been used for unsupervised training, e.g., Fukushima [Fuk75] [Fuk80], Kohonen [Koh82] and Amari [Ama83]. The term is used to describe any method where outputs compete for an input before weight modification.

6.4.4. Feedforward Networks

Several types of feedforward network (perceptrons, Adalines, Madalines, multilayer perceptrons and cascade correlation networks) were described in the previous section. This section reviews some of the other feedforward network structures that are suitable for character recognition.

Kohonen's *learning vector quantization* (LVQ) [Koh90a] [Koh90b] [Koh90c] is a nearest-neighbour classifier similar to the statistical nearest-neighbour classifiers (see section 6.2.2). Input vectors are matched to one of a set of exemplar vectors according to a distance metric (usually the shortest Euclidean distance between vectors). The difference with the LVQ classifier is that it uses a clustering procedure to reduce the number of exemplar vectors (called *codebook vectors*) required to represent the classes. This is very useful in real applications like character recognition where the large training sets required make nearest-neighbour matching too slow for practical use.

Vector quantization (sometimes known as *block quantization* or *pattern-matching quantization*) is a vector reduction technique which has previously been applied to many data compression applications, such as image compression (Gersho and Ramamurthi [GR82]) and speech coding (Makhoul *et al.* [MRG85]). A comprehensive survey of these applications can be found in Gray [Gra84]. In recent years, adaptive vector quantization has been formulated as a neural network (where the values of the vectors form the network weights) by Kohonen and has become a popular neural paradigm.

Training the classifier involves an initial determination of how many codebook vectors are to be used and how they are to be distributed among the classes. Kohonen recommends that approximately the same number of codebook vectors are assigned to each class. The total number of codebook vectors is limited by the required recognition speed and therefore by the computational power of the specific system. The initial values of the codebook vectors are selected from the training set by finding samples of the intended classes. They are first checked with a tentative k-nearest-neighbour classification using the other training samples to ensure that they lie roughly within their respective class boundaries. Only samples which are correctly classified on this way are accepted.

The learning rules are iterative error-correcting procedures. They cycle through each sample in the training set and shift the positions of the closest codebook vectors towards or away from the sample according to whether they classify it correctly or not. Kohonen describes several LVQ training procedures in his 1992 LVQ Program Package [KKL92]. Learning usually begins with an optimized-learning-rate LVQ [KKL92], where each codebook vector has an individual associated learning rate. Optimal values for the learning rates can be determined which allow the classifier to rapidly converge to its asymptotic recognition accuracy. Improvements to the accuracy can be achieved by fine tuning the classifier with alternative LVQ rules [Koh90a] [Koh90b] [Koh90c]. The aim of learning with these algorithms is not to approximate the class density functions (as in Makhoul *et al.* [MRG85]) but to directly define the class boundaries under nearest-neighbour classification [KKL92].

An alternative to the iterative training strategy is a batch training procedure. Batch clustering algorithms are sometimes applied though they have a tendency to become stuck in local minima of the classifier's energy function. The most common of these procedures is the Linde-Buzo-Gray algorithm [LBG80] which is equivalent to the *k*-means algorithm (see section 6.2.3).

LVQ is simple and powerful, giving greater accuracy than a parametric Bayesian classifier (see section 6.2.1) [Koh90b]. Its main drawback is its long training time. Fine tuning is a very lengthy process but the improvement in accuracy is worthwhile. The speed of the nearest-neighbour operation is also a limiting factor but the vector reduction makes LVQ much faster than a conventional nearest-neighbour classifier. The standard LVQ classifier does not produce any certainty or confidence factor of its output. A confidence measure can be obtained by formulating them as radial basis functions, as has been done in chapter 7.

Radial basis function networks (RBFs) (Broomhead and Lowe [BL88], Moody and Darken [MD89]) are formulated as single-layer perceptron networks, but rather than multiplying the inputs by the weights, the distance (usually Euclidean) is taken between the input vector and the weight vector. This is then passed through a radial basis activation function which produces a maximum at zero and approaches zero at infinity. The neuron with the closest stored weight vector to the input scores highest. RBFs therefore perform a nearest-neighbour classification. As in LVQ, a reduced set of stored vectors is trained to represent a larger sample set by a clustering procedure. Methods of training RBF classifiers are described in detail by Musavi *et al* [MAC92].

Some RBFs include additional weights on the vectors so that a weighted nearest-neighbour classification is obtained. Whereas codebook vectors in LVQ classifiers represent hyperspherical distributions of the training vectors, in RBF classifiers they can represent hyperellipsoidal distributions. A further extension assumes a Gaussian distribution of the classes and models this by estimating the covariance matrices for each codebook. The more powerful decision regions that can be formed with the hyperellipsoidal representations offer the possibility of improved accuracy over learning vector quantization. However, the increased complexity of the classification also increases the required training time.

Albus's *cerebellar model articulation controller* (CMAC) [Alb75a] [Alb75b] is a feedforward model of neuronal structure based on the cerebellum which controls motor processes in the brain. The original method has been developed by a number of researchers, among them Forsyth [For90] who applied hashing functions to reduce the CMAC's large memory requirements and Cornforth [Cor93] who added probabilistic class output.

Continuous inputs are grouped together by quantizing bands of hashing functions which divide the input space into hypercubes. The bands, and therefore the hypercubes, overlap. The hashing functions address a lookup table which contains real-valued numbers that are set during training. Because the hypercubes overlap, each input lies in the range of several hashing functions and therefore generates several addresses in the table. Addressed values in the lookup table are added to give the output value of the classifier. Different output classes are represented by points on a scalar range. Whichever class is closest to the output value is taken to be the classification. This can be seen as a discrete variant of a radial basis function [Sar94].

To give a measure of the probability of a classification, the distance to the output point from each class point can be used. This is a limited solution, however, and a better method is to use a different lookup table for each output class [Cor93]. The output value then becomes a vector, with one axis for each class, and the magnitude along each axis indicates the probability of the associated class. This increases the, already large, memory requirements considerably.

The use of lookup tables rather than the solution of equations makes the CMAC very fast in operation. Learning is also fast compared to many other neural classifiers, although Albus's original algorithm was sometimes slow to converge. Methods have been developed to speed up CMAC learning, e.g., Parks and Militzer [PM89] and Cornforth [Cor93], who proposed a single-pass training algorithm. The class boundaries formed are arbitrarily complex and so CMAC can learn nonlinearly separable classifications. Its main disadvantage is the large amount of memory that must be addressed by the hypercubes. Although the use of hashing reduces this memory requirement [For90] it is still probably the largest of any classification technique.

CMAC was developed for robot control but has gained popularity as an alternative to back-propagation in several other applications. Miller *et al.* [MGK90] used CMAC in a simple character recognition experiment. The application to real character recognition is limited by the huge memory requirements when large sets of classes are involved.

Yair and Gersho [YG90a] developed a deterministic, feedforward version of the Boltzmann machine (see section 6.4.5) which they called the *Boltzmann perceptron network* due to its multilayer perceptron-like structure. It is still a probabilistic model — there is an underlying stochastic network from which it computes statistical averages of the outputs.

No simulated annealing is required and learning is similar to back-propagation. A method called partial conjugate gradient search (Luenberger [Lue84]) is used in minimizing the network's energy cost function. Learning is faster than in a Boltzmann machine.

The Boltzmann perceptron network (BPN) evaluates *a posteriori* class probabilities as do Bayesian classifiers (see section 6.3.1) but without making assumptions about the underlying probability distribution (Yair and Gersho [YG90b]). Yair and Gersho's results show that performance of the BPN is comparable to a Bayesian classifier.

While the BPN is simpler than the Boltzmann machine it is still quite complex. Despite this complexity it does not seem to give any real advantage over Wright's probabilistic neural networks [Wri89] (see section 6.4.3).

Hecht-Nielsen's *Avalanche Matched Filter* (AMF) [Hec82] [Hec87a] learns spatiotemporal patterns of continuous activation functions using Hebbian learning. Two weight matrices are trained; one learns the spatial patterns and the other learns the temporal relationships. These networks are fast and can learn arbitrary continuous spatiotemporal patterns relatively easily. The main disadvantage of AMFs is that a separate network is required for each sequence to be learned. AMFs are suitable for recognition tasks where patterns are temporally ordered. They can be used for certain aspects of character recognition, such as word recognition where the transition probabilities between characters could be assigned to the temporal weight matrix. A similar approach was used for speech recognition by Lippman and Gold [LG87] in an extension of the AMF which they called a Viterbi Net.

There are many other feedforward networks which can be trained by supervised learning procedures but not all of them have been applied to character recognition. Simpson [Sim90] gives good descriptions of most of the main alternatives and discusses the applications for which they are suitable.

Unsupervised learning in feedforward networks mainly uses matrix formulations of linear associative memories. *Linear associative memory* (LAM) (Anderson [And68]) stores pattern pairs, $(\mathbf{a}^{(p)}, \mathbf{b}^{(p)})$, in a correlation weight matrix:

$$w_{ij} = \sum_p a_i^{(p)} b_j^{(p)}$$

The attraction of LAMs is that if the stimulus vectors are mutually orthogonal then recall is perfect, otherwise there will be some crosstalk from other memorized response vectors in addition to the associated response [WZ89]. They also learn quickly as examples do not have to be presented iteratively.

Wee [Wee68] trained LAMs using the Moore-Penrose pseudo matrix inverse [Pen55] [Pen56] to optimize the Least Mean Square error. Kohonen and Ruohonen [KR73] independently produced the same solution, using Greville's algorithm [Gre60] to construct the pseudoinverse incrementally in an efficient and simple manner. This solution is known as the *optimal linear associative memory* (OLAM). OLAMs have been applied to character recognition by Wee [Wee70], and Chieuh and Goodman [CG88].

Steinbuch's *learning matrix* [Ste61] was an early unsupervised feedforward network model. Although it is rarely used today, it has historical significance as it was the precursor of competitive learning. All these neural memory models can be formulated as Hebbian learning systems [Car89].

Fukushima's *neocognitron* [Fuk80] [Fuk88] [Fuk89] is a hierarchical neural network designed for character recognition. It does not require prior feature extraction, instead it takes a bitmap as input. Unsupervised training of the neocognitron is performed by a competitive, self-organizing mechanism during which the network's neurons learn to detect features of the inputs. The network is based on the human visual system and as such is hierarchical in nature. Low levels of the system detect simple features and higher levels extract more complicated ones from the lower levels.

The competitive learning procedure adapts weights by a variant of Hebbian correction. The highest responding neuron in a layer, called the *seed cell*, has its incoming connection from an input reinforced if the input is also responding (non-zero). Input connections to the seed cell's neighbouring neurons are also modified so that they learn to detect the same feature. The neighbouring neurons correspond to slightly different locations in the preceding layer so this modification results in regions of layers which detect identical features in different positions. This architecture is intended to make the recognition invariant to shifts in position of the target image. Fukushima's earlier *cognitron* network [Fuk75] had promising pattern recognition ability but could only recognize patterns in the position in which they were learned. The redundancy of feature detectors also aims to make the recognition invariant to size and partially invariant to deformation.

Barnard and Casasent [BC90] have shown that the neocognitron is *not* intrinsically invariant to positional changes and that its shift invariance capability depends on the choice of certain parameters. They also provide equations for selecting these parameters so as to give approximate shift invariance. There is a trade off, however, between the neocognitron's sensitivity to differences in its inputs and its tolerance to shifts in image location.

A size and translation invariant recognizer for character bitmaps is potentially a very useful classifier. However, the neocognitron requires a huge number of neurons. There is a large degree of redundancy as most of the neurons are involved solely in trying to achieve shift invariance. Also, real problems require a large number of layers. This makes the network impractically slow. Fukushima [Fuk80] [Fuk88] has demonstrated the neocognitron's ability to learn small sets of good quality, 16x16 pixel, digit bitmaps but it seems unlikely that the network will be practical for a full system (digits and letters) with realistic levels of noise.

An interesting development of the neocognitron includes feedback connections and operates as a position and deformation invariant associative memory. Fukushima [Fuk86b] used this to recognize pairs of digits by a variant of recursive segmentation-classification. The network was able to accurately separate and recognize a number of examples.

Other feedforward network models for unsupervised learning that have been used for character recognition include self-organizing maps and counterpropagation networks.

Kohonen's *self-organizing maps* [Koh82] [Koh84] [Koh90b], also called *topology-preserving maps*, are the unsupervised forms of learning vector quantization (see above). The self-organizing map uses a competitive learning mechanism to distribute a set of unlabeled codebook vectors such that they cover the area spanned by the set of training vectors. Classification of input vectors is then performed by nearest-neighbour matching of the codebook. Kohonen showed that after self-organization is complete, each codebook vector is at the centre of a decision region [Koh86]. The decision regions form a Voronoi tessellation, a tiling of the input space where the decision boundaries are perpendicular to the lines joining the centroids. A number of alternative competitive learning algorithms for unsupervised vector quantization are described and compared in Ahalt *et al.* [AKC90]. *Counterpropagation* networks (Hecht-Nielsen [Hec87b]), are an extension of self-organizing maps with an additional layer which allows them to encode a response vector as well as the stimulus vector, i.e. they are heteroassociative models rather than autoassociative.

Unsupervised learning is less common in character recognition applications as the classes of training sample characters are usually known. Simpler and more accurate supervised learning is generally preferred.

6.4.5. Feedback Networks

One of the most popular feedback networks for character recognition is the *Hopfield net* (Hopfield [Hop82], Hopfield and Tank [HT85]). This is a single-layer feedback network that is fully connected, i.e., each neuron is connected to every other neuron. Hopfield nets can be used as autoassociative memories or to solve optimization problems. There are both discrete (binary) and continuous (analogue) versions. The discrete version is also known as the *discrete autocorrelator.*

Discrete Hopfield nets are trained by an unsupervised Hebbian learning method. A correlation weight matrix is generated, as for linear associative memories (see section 6.4.4). Correlating one exemplar with another gives a first order encoding which can only perform linearly separable mappings. Higher order encodings are obtained by correlating each exemplar with pairs of other exemplars. This enables the net to learn nonlinearly separable functions. Continuous Hopfield nets cannot be encoded in such a simple way and are trained by alternative techniques such as competitive learning or signal Hebbian learning.

Recall operates by placing the input values on the network's outputs and then repeatedly updating the neuron outputs. The new neuron outputs are calculated as in the standard neuron model. The discrete Hopfield net uses a threshold activation function; the continuous net uses a sigmoid function. Updating is usually by a random or sequential, iterative sequence, but can also be a batch process where all the new neuron activations are calculated before any of the outputs are updated. Continuous nets use the iterative method and are highly asynchronous as they can also allow propagation delays between neurons. The updating causes the output neuron values to change but they eventually settle at stable values. When all the neuron values stop changing, they are output from the net.

The main question concerning feedback networks is whether or not they will converge to a stable state for all inputs. Hopfield nets are symmetric: the weight from neuron i to neuron j is equal to the weight from neuron j to neuron i. Hopfield [Hop82] showed that in discrete symmetric networks the system will always converge to a minima of the network's energy function, though not necessarily the global minimum. He also gave evidence for symmetric networks occurring in nature. Similar convergence properties can be proved for continuous Hopfield nets [Sim90].

Hopfield nets are particularly useful because they are able to recreate patterns from partial inputs and are tolerant to noise. Their main limitation is the comparatively small number of exemplars that can be accurately stored. Amari and Maginu [AM88] showed that the discrete Hopfield net can perfectly recall up to $n/(2\ln n + \ln\ln n)$ exemplars, where n is the number of neurons in the net. Therefore as n increases the relative increase in the number of perfectly stored exemplars decreases.

Hopfield nets have been studied and modified by many researchers. A comprehensive listing of this work can be found in Simpson [Sim90]. Hopfield nets have been applied to the recognition of bitmap characters by several researchers, e.g., Chieuh and Goodman [CG88], Pawlicki *et al.* [PLH88], and Schalkoff [Sch89].

A heteroassociative version of Hopfield nets was introduced by Soffer *et al.* [SDO86] [SMO86] and refined by Kosko [Kos88] as the *bidirectional associative memory* (BAM). Two layers are used, with the neurons in one layer corresponding to the stimulus pattern and those of the other layer corresponding to the response pattern. Unsupervised learning and recall operates in the same way as for Hopfield nets and similar convergence properties have been proved (Kosko [Kos88]). There are both discrete and continuous versions (the continuous version is known as *adaptive bidirectional associative memory* (ABAM) (Kosko [Kos87])). A version of ABAM which uses competitive learning is called the *competitive ABAM*.

The BAM and its variants have very limited storage capacity and are unlikely to have practical applications in character recognition. However, they have very good noise tolerance abilities and may be useful in a limited way.

The main paradigm for supervised learning in feedback networks is the *Boltzmann machine* (Hinton *et al.* [HAS84]). The Boltzmann machine is a symmetric network similar to the one developed by Hopfield. It uses stochastic neurons, instead of the deterministic neurons of Hopfield nets, which introduce a probabilistic element into the system.

Boltzmann machines use a combination of Hebbian and stochastic learning. There are two stages to learning: a training phase and a testing phase. Weights are modified by Hebbian learning during the training phase and by an anti-Hebbian rule (negative modification) during testing [Sej88]. The stochastic part of the learning process is as described in section 6.4.3. The non-improving weight changes are accepted probabilistically according to the Boltzmann distribution, from which the classifier gets its name. The learning procedure minimizes a cross-entropy cost function called the *asymmetric divergence* — the divergence between the actual and desired conditional probability distributions of the output neurons.

The Boltzmann machine has performed well on some difficult problems, e.g., learning symmetry [SKH86], and has been shown to form useful internal representations of its inputs [AHS85]. Unfortunately, learning is extremely slow because probabilities have to be estimated. It is also limited in that it uses discrete binary activations of its neurons. Its application to character recognition has been minimal, although digit recognition has been attempted (Pawlicki *et al.* [PLH88]). Kuner [Kun89] developed a method for matching attributed and non-attributed relational graphs using Boltzmann machines. This could be used for structural character recognition.

A similar classifier is the Cauchy machine (Szu [Szu86]) which replaces the Boltzmann distribution with the Cauchy distribution to allow a faster simulated annealing process.

Another feedback paradigm for unsupervised learning is Grossberg's *adaptive resonance theory* (ART) [Gro76b]. This is a development of the earlier additive and shunting Grossberg networks that were described in section 6.4.3 and is based directly on neurophysiology. There are two main forms of ART, a *binary adaptive resonance theory*, known as ART1, and *analogue adaptive resonance theory*, known as ART2.

ART1 (Carpenter and Grossberg [CG87a]) is a two-layer competitive learning model similar to that described in section 6.4.3. Recall is by on-centre/off-surround competition, using the shunting Grossberg method with gain control to amplify signal and suppress noise (see section 6.4.3). The important feature of the ART1 network is its feedback connections from the second layer to the first layer. After a second-layer winner is chosen by the competitive recall, it sends a signal back to the first layer via these feedback connections. This allows the original input activations to be compared to the feedback pattern.

The difference between the input and the feedback is compared to a threshold *vigilance parameter*. If the difference is less than the vigilance parameter then the input and the stored pattern of the winner neuron are said to *resonate*. The weights to the winner neuron are then modified. If they do not resonate then the input is considered to be too far from the stored pattern of the winner neuron. Instead of modifying the weights to the winner, the input is encoded onto an uncommitted second-layer neuron. The vigilance parameter therefore determines the degree of separability of the classes. Lippman [Lip87] pointed out that ART1 learning is basically a clustering method, very similar to Hartigan's sequential leader clustering algorithm [Har75].

Weight modification and encoding of inputs onto output neurons is achieved by either 'slow' or 'fast' learning. The 'slow' learning methods are variants of the gated delay long term memory equation (see section 6.4.3). 'Fast' learning is a much quicker but less statistically representative encoding of the input, based on the correlation between the original input and the feedback. The details of these learning methods are described in [CG87a] and [Sim90].

The type and rate of learning relates to the stability-plasticity dilemma of Grossberg's earlier networks. If the learning is too slow, then it is not adaptable enough to learn new patterns (it is too 'stable'). If learning is too fast it can cause previously learned patterns to be overwritten (it is overly 'plastic'). Carpenter and Grossberg designed the ART network to allow the dilemma to be resolved [Car89]. The use of the feedback and vigilance parameter enables the previously learned patterns to influence learning, thereby allowing a reasonable degree of plasticity whilst keeping a check on the stability of old patterns.

The main limitation of ART1 is that it only has binary inputs and outputs. While this is sufficient for the recognition of bitmaps, continuous-valued inputs and outputs allow more general application to pattern recognition problems. Another problem is that learning tends to generate too many clusters. If the vigilance parameter is set too low, it can rapidly use up all the second-layer neurons. Carpenter and Grossberg [CG87b] developed the analogue version, ART2, to tackle these problems. ART2 operates and learns in essentially the same way as ART1 but has continuous-valued inputs and outputs. Each input-layer neuron contains a complicated three-layer feedback system which is used to normalize the inputs. This normalization helps reduce the number of clusters created. Another way of reducing the cluster proliferation is to adapt the vigilance parameter during learning to control the cluster sizes [CG87a].

The strengths of the ART networks are their large storage capacity and ability to encode patterns in clusters of varying complexity. ART is a powerful classification paradigm for real-world problems with large numbers of complex classes. The on-line learning mechanism is also useful for character recognition as it allows the network to learn new users' handwriting while the system is in use.

ART networks have been applied to character recognition in several experiments. For example, Gan and Lua [GL92] used them to classify a database of 3755 Chinese characters with high accuracy. Sulaiman and Evans [SE95] used character recognition experiments to compare ART with multilayer perceptrons, counterpropagation networks and self-organizing maps. Srinivasa and Jouaneh [SJ92] used Widrow's Madaline networks [WW88] to perform size, position, orientation and slant normalization of character bitmaps and then classified the normalized characters using an ART1 network. They later merged the normalization process into the ART1 architecture by adding an additional layer [SJ93].

6.5. Remaining Problems — The Need for Context

We have seen how classifier design has advanced to the stage where classifiers can be trained to learn arbitrary decision boundaries. Whilst these classifiers perform with very high accuracy on the training sets, their error rates on unseen data are still too high for use in a practical character recognition system. While much research continues to develop classifiers in the hope of reducing this error rate, it is unlikely that any great improvement in generalization can be made. The problem with most classification tasks being attempted is not that the basic classifier is essentially poor, but that the classifier does not receive enough information to disambiguate its failure cases.

In almost all classification problems the addition of contextual information will improve the correct classification rate. Machine classification has yet to rival human experts because of the wide range of high-level knowledge available to the human brain [RM86]. Incorporating this knowledge should be a major aim of artificial intelligence research, particularly so in the area of character recognition.

The usual cause of generalization errors is that the attempted classification is based on input data from an isolated occurrence of the problem but the input data alone is not sufficient to disambiguate cases. Much research has been carried out on selecting features which reduce this occurrence [SS98] but no choice of features is likely to be perfect. Class boundaries for real world problems generally overlap. While the specific class memberships of the training set can be learned, this does not overcome the problem that, in the general case, the class of some inputs may be ambiguous. Regardless of how good the feature set may be, a classifier cannot distinguish the classes using only the features of a single case. This is certainly the case in handwritten character recognition. In most examples of handwritten script, different characters are often identical in shape, '5's and 'S's for example, or '2's and 'z's. What might be an 'r' in one person's writing might look like a 'v' in someone else's. It is therefore impossible to correctly identify the characters in isolation.

Human performance at isolated character recognition corroborates this. Edelman *et al.* [EUF90] report that, in the absence of context, humans correctly recognize:

> "96.8% of handprinted characters [Neisser and Weene 1960], 95.6% of discretized handwriting [Suen 1983] and about 72% of cursive strings (see [Edelman 1988] appendix 1)."

Suen [SSK77] additionally reports a 4% human error rate on handprinted characters without context. It is generally accepted that these human errors are almost entirely caused by ambiguous characters. This ambiguity is caused either by corruption of the image by noise, or by natural class overlap. The human error therefore closely approximates the amount of character class overlap in real applications at the underlying image level (rather than at the feature level), which puts a practical limit on noncontextual generalization accuracy for both humans and machines.

To overcome this limit therefore requires more information to disambiguate cases in overlapping class regions. This often means looking at other cases in the input, usually those immediately preceding and/or succeeding the original case. In character recognition these surrounding cases, when grouped together, form words or sentences. By looking for contextual sensibility across groups of cases it is usually possible to disambiguate the original target.

The following section reviews methods for contextual processing of characters.

6.6. Contextual Processing

Approaches to contextual processing in character recognition are almost all based on word level checking of spelling. Higher level grammatical and semantic context has rarely been applied. Several of the contextual techniques applied to character recognition were originally developed for speech recognition, which is very similar to cursive script recognition at the higher level stages. Reviews of contextual processing for character recognition have been published by Toussaint [Tou78], and Elliman and Lancaster [EL90].

Processing takes one of two forms. In one case the context is used for verification of the bottom level classification. Isolated character classes are assembled into words or sentences and the contextual processing determines whether or not they are valid. Invalid words or sentences may be rejected. In the second case, the processing is used to correct recognition and spelling errors. Words or sentences are matched against legal patterns and the most probable ones are output, often with an associated confidence measure. The second approach is much more useful and more common.

Contextual mechanisms rely on useful representations of contextual information, e.g., valid words and grammatical models. These are established in advance, usually by obtaining a machine-readable dictionary or by analysing sample texts from the language. Some systems can also learn new words from the source text while they are in use.

Good representations are difficult to obtain. There are few machine-readable dictionaries available and most of them are limited in that they only list the simplest form of each word. For example, the word "camp" may be present in the dictionary but "camps," "camped" and "camping" are not. The various endings must be generated by linguistic rules. Most dictionaries do not rate words according to their frequency so representations of many rare words can result. It is even harder to represent grammatical models of language which cover every legal sentence.

Where character recognition is to be used for a particular application, it is preferable to take a smaller, application-specific dictionary [Sue79]. These have the advantage of being much faster to search with dictionary matching methods, and give a more accurate representation of the desired context. However, for a general character recognizer, the problems of obtaining good context representations from large dictionaries must still be overcome.

6.6.1. Dictionary Methods

The simplest and most obvious application of contextual processing to character recognition is the matching of words against a dictionary (sometimes called a *lexicon*). This is basically a string matching operation. String matching for isolated character classification has previously been discussed in section 6.3.2. The methods for string matching of dictionaries are generally the same.

The most common approaches determine similarity of strings according to the Levenshtein distance [Lev66] or edit distance (see section 6.3.2), e.g., Okuda *et al* [OTK76]. Almost all methods use approximate string matching where substitution, insertion and deletion errors are allowed. This is far better than exact matching since incorrect character classifications are to be expected. Allowing for insertions and deletions can compensate for segmentation errors where a character is split into more than one segment or multiple characters are merged into a single segment.

Takahashi *et al.* [TIA90] applied a strategy for spelling correction to character recognition which selected a reduced candidate set from a dictionary by looking for words containing the rarest letters in the input word. The dictionary was ordered so that the least frequent letters of each word occur first. They suggested several string similarity measures for choosing the correct word from the reduced candidate set including Levenshtein distance and Hunt and Szymanski's Longest Common Subsequence (the longest subsequence of letters which occurs in both strings) [HS77]. They suggested that most errors in the recognition of machine-printed characters are caused by incorrect classification rather than segmentation. To simplify the comparison they allow insertions or deletions, but not both. They achieved high word recognition rates (≈90%) on reasonable quality printed text but it is unlikely that their method would work as well on handwritten characters where segmentation errors are a much greater factor in character mis-classification.

Another method of string matching is probabilistic matching (Hall and Dowling [HD80], Kashyap and Oommen [KO84]). This formulates the problem of determining string similarity in terms of the probability, $P(X,Y)$, that a string, Y, is the correct string when X is the observed string. String matching becomes a task of finding the maximum value of $P(X,Y)$ over all X. The task is broken down in a similar way to edit distance matching. Probabilities are determined for single character substitution, insertion and deletion operations which turn one string into the other. The product of the individual probabilities gives the total probability, $P(X,Y)$, for the whole string.

Kashyap and Oommen [KO84] applied their algorithm to text correction. They claim that the probabilistic approach performs better than the distance-based approach, which is more prone to error on short words. Bozinovic and Srihari [BS82] applied a probabilistic string matching method to the contextual postprocessing of cursive script. They tested the method on strings with simulated segmentation and recognition errors. It performed well on long words, but unlike Kashyap and Oommen they found that errors increased on shorter words. The fact that shorter words present a smaller context suggests that both the probabilistic and distance-based approaches will be less accurate on them.

An important factor in string matching of large dictionaries is the way in which the dictionary is stored. A simple list of words is very slow to search. If the word is not present then the process has to check every word in the dictionary and never finds a match. Fredkin [Fre60] proposed trie memory which takes advantage of the redundancies (common subsequences of letters) in dictionaries. Trie memory can be searched much faster than lists, most importantly when the target word is not present in the dictionary. Its main disadvantage is its inefficient storage of small dictionaries but this inefficiency reduces for large stores. A detailed analysis of the efficiency of trie storage in relation to the size of dictionary is given by Sinha [Sin87b].

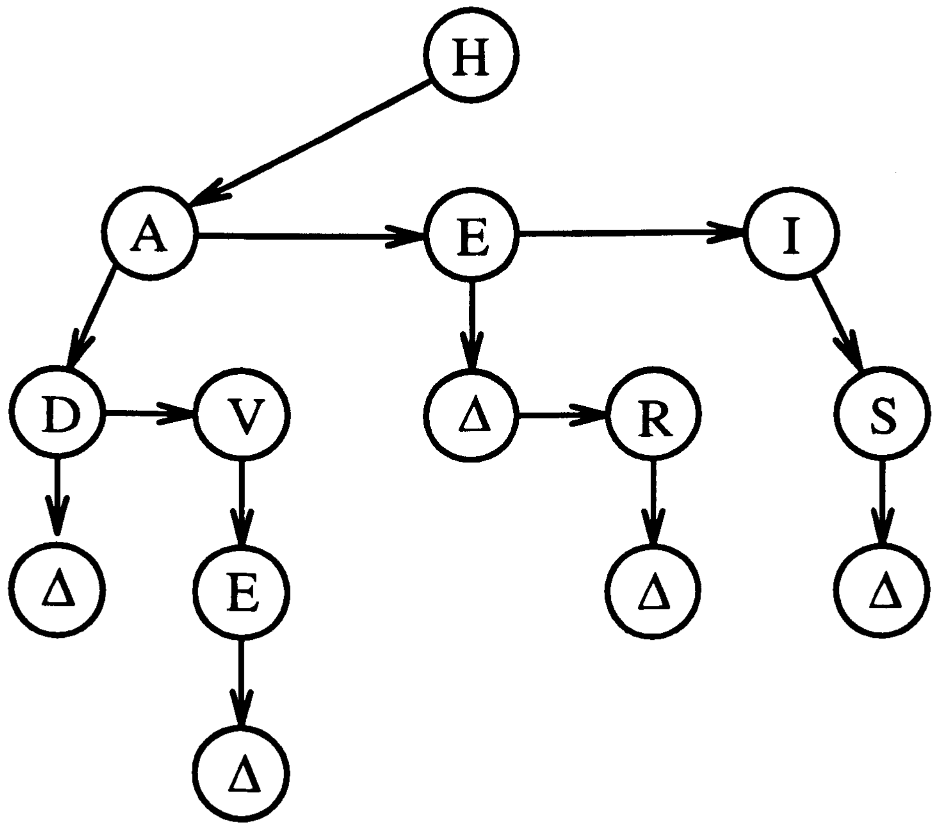


Figure 6.4 Example of trie memory represented as a binary tree, after Knuth [Knu73]

Fredman's trie was originally represented in a table of registers. It is more intuitively depicted as a linked list or binary tree structure (Knuth [Knu73]). Figure 6.4 shows a simple binary tree representation of a trie for the words HAD, HAVE, HE, HER and HIS. The trie is searched by following the right hand branches until a match is found with the symbol at a node, then the left hand branch is followed. Δ indicates the end of a word. Tries have been used for text correction by several researchers, e.g., Muth and Tharp [MT77], Hull *et al.* [HSC83] [SHC83], and Downton and Tregidgo [DT91].

Another strategy for dictionary storage with fast access is to order words by frequency so that commonly occurring words are compared first. The classical method for this is the *frequency ordered binary search tree* (see Knuth [Knu73]). This arranges the tree so that each node's branches partition its descendants on either side of the node's lexical (alphabetical) value. Although the nodes are frequency ordered, this can produce uneven trees which are not optimal for searching. Sheil [She78] presented the *median split tree* which partitions each node's branches on either side of the median lexical value of the node's descendents. This gives a much more balanced tree and faster search times.

Kohonen and Rehkala [KR78] used redundant hash addressing to find indices to the dictionary. "Features" of the target words are extracted. These are *n*-grams, strings of *n* consecutive letters. The *n*-grams are then encoded by a hashing function to produce an address into a table of pointers to words in the dictionary. The addressed pointers should indicate words which have similar "features" to the target word. Matching is therefore speeded up by beginning the search with words which closely match the input. Hashing was also used by Doster [Dos77].

A less effective but quite common strategy is to divide the dictionary to be searched into several subdictionaries according to the length of word, e.g., Bledsoe and Browning [BB59], Shinghal and Toussaint [ST79b], Burr [Bur87]. A target word need only be matched against the list of words with the same length. Although this gives an improvement in speed, it does not allow for approximate string matching with deletions and insertions.

String matching methods have been shown to perform well on long words but are less accurate on short words [EL90]. Short words often have greater similarities and are less easy to disambiguate. Verification methods are limited to simple applications where high accuracy recognition of the isolated characters is possible. In real applications it is very likely that errors will occur, which means that many words will not be validated. Methods which correct approximately matched words are more appropriate and have been more successful.

The main problem with dictionary matching methods is their computational complexity. Searching large dictionaries, even with special memory structures such as tries, is too slow for a practical system. At current processor speeds, dictionary methods are only suitable for applications where small dictionaries can be used to store contextual domain knowledge.

6.6.2. Markov Methods

For general applications, where words may come from large vocabularies, an alternative to dictionary matching is popular. Rather than attempting to match entire words, the language is modeled as a Markov process. Markov models, also called Markov sequences or Markov chains, represent letters in the words as an ordered set of states. The transitions from one state to the next are probabilistically dependent on one or more of the preceding states. These processes are represented by assigning transition probabilities to *n*-grams (sequences of *n* letters). Usually bigrams (pairs of letters), sometimes called digrams, or trigrams (triples) are used (e.g., Raviv [Rav67], Donaldson and Toussaint [DT70], Neuhoff [Neu75] and Shinghal *et al.* [SRT78]) but the more computationally expensive quadgrams have also been considered (e.g., Carlson [Car66]). Shinghal *et al.* [SRT78] extended the transition probabilities to depend also on the position the *n*-grams occur in the word. These probabilities are usually estimated from a large sample of text, known as a *corpus*.

The recognizer produces candidate sets of letters for each position in the word, along with confidence factors. Matching of the input word involves finding the most probable sequence of candidate letters (states) according to the *a priori* probabilities and confidences of the letters. Note that the *a priori* probabilities of words are determined by the Markov model and are not related to their frequency of use. There are several methods for maximizing the probability of a word. The main ones are the *modified Viterbi algorithm*, *recursive Bayes algorithm* and *probabilistic relaxation*.

Viterbi's original algorithm [Vit67] was developed for use in cryptanalysis to find maximum likelihood estimations of sequences. This was extended by Forney [For73] to estimate maximum *a posteriori* probabilities of words for use as a contextual postprocessor in character recognition. The basic Viterbi algorithm considers all of the possible states (usually 26 states, one for each letter of the alphabet) as candidates for each position in the word. This exhaustive procedure is computationally expensive and a faster modification was presented by Shinghal and Toussaint [ST79a] which uses a reduced candidate set for each position, consisting of the first *d* most probable characters indicated by the classifier. Shinghal *et al.* [SRT78] suggest that the optimal value of *d* is five. The Viterbi algorithm and its modification are simple and accurate and have become the most popular methods for sequence estimation in Markov models of text. They have been applied to character recognition by many other researchers, including Neuhoff [Neu75], Hull and Srihari [HS82], Sinha *et al.* [SPH93], and Kopec and Chou [KC94].

Raviv [Rav67] proposed the recursive Bayes algorithm as an optimal method for compound decision making in Markov sequences. Raviv applied the method to contextual postprocessing of machine-printed characters. The algorithm is accurate but is quite slow due to its complexity and recursive nature. Shingal *et al.* [SRT78] showed that a heuristic approximation to the recursive Bayes algorithm achieved comparable results to the complete algorithm at much faster speeds. Similar applications of compound decision theory to character classification in Markov models were described by Abend [Abe68] and Duda and Hart [DH68].

Goshtasby and Ehrich [GE88] applied the relaxation process of Rosenfeld *et al.* [RHZ76] (see section 6.3.4) to contextual processing of characters. The probabilistic relaxation labeling process iteratively modifies the probabilities of characters according to the states of their neighbours and the transition probabilities between them. It operates both left to right and right to left, whereas a conventional Markov process only propagates information from left to right. The iterative process eventually converges to a state where only one character for each position has a probability of 1.0. Goshtasby and Ehrich used the method for contextual postprocessing of low resolution, lower case, machine-printed characters.

The advantage of n-gram techniques are that they do not need to store entire large dictionaries. However, the number of different letter combinations occurring in a corpus grows rapidly as n increases. Suen [Sue79] conducted an extensive study of n-grams of the English language. He found that for $n \geq 4$ the number of combinations occurring in a reasonable sized corpus is so large that it requires more memory to store the n-grams than to store the words from which they are derived. In any case, the number of letter combinations for which transition probabilities must be stored (including those that do not occur in the corpus) is very large. Assuming the states are the 26 letters of the alphabet, there are 676 bigrams, 17576 trigrams and 456976 quadgrams. Quadgrams and higher order Markov dependencies are therefore rarely used.

Markov methods are good at correcting local errors but their lack of global knowledge makes them less effective than dictionary look-up techniques. Although they are flexible enough to recognize or correct words which do not appear in a standard dictionary but follow the underlying model of the language, they often produce such words in place of standard dictionary words. Also they only correct substitution errors, whereas dictionary methods can generally correct insertion and deletion errors as well.

Vossler and Branston [VB64] compared dictionary look-up with a bigram technique. Using a 364-word dictionary they found the dictionary method was far superior, with a correction rate of 93%, compared to only 45% for the bigram method. Hanson *et al.* [HRF76] also found that bigram and trigram techniques had limited effect on the word error rate and could actually increase the character error rate by creating long runs of erroneous characters. The reason for this is that once an error occurs it effects the following character, which in turn effects the next one, and so on, so that further errors are more likely. They conclude that "this is an ineffective use of contextual information. ... We do not believe that simple and efficient techniques under assumptions of Markov dependence will result in highly significant reductions in the error rate."

6.6.3. Hybrid Methods

A number of methods have attempted to combine the more accurate dictionary methods with the faster Markov approaches in order to gain the benefits of both. The Markov (statistical) methods are sometimes called *bottom-up* or *data driven* approaches. The dictionary matching methods are called *top-down* or *concept driven* approaches. The hybrid methods are therefore also known as *bottom-up and top-down* methods [ST79b].

An early hybrid method was devised by Riseman and Ehrich [RE71] to address one of the main problems with dictionary matching, namely that even with only a few candidate characters for each position, there are a great number of candidate sequences. Duda and Hart [DH68] pointed out that for a ten-letter word with four possibilities in each position there are over a million different combinations. Matching each of these against a dictionary is very slow. Riseman and Ehrich proposed a variant bigram technique to rapidly remove illegal combinations from consideration. This was generalized to n-grams by Riseman and Hanson [RH74].

Their approach starts by considerably reducing the storage requirements of n-gram techniques. Rather than store transition probabilities for each n-tuple of letters, their *binary n-grams* store either 0 or 1 according to whether the transition probability is 0 or nonzero, respectively. This reduction of storage requirements allows them to store much more information about n-tuples in the underlying language model. They store *positional binary n-grams.* For simplicity of explanation let us assume they are binary digrams. One binary digram is stored for each possible pair of letter positions in a word. The positional binary digram D_{ij} is a 26x26 binary matrix indicating the allowable (value 1) and illegal (value 0) pairings of letters in positions i and j in a word, where $1 \le i \le j \le n$, and n is the length of the word. The pairings of letters do not have to be contiguous as in the standard Markov model; therefore a word of length n requires $\left[\begin{matrix} n \\ 2 \end{matrix} \right]$ binary digrams. By checking candidate letter sequences against this model, sequences with pairs of letters in illegal positions are quickly rejected. The remaining allowable candidate sequences are then checked against the dictionary. The method greatly reduces the number of candidate sequences that have to be matched. Binary n-grams have also been used by Hull and Srihari [HS82].

Another early hybrid method is known as the *Predictor-Corrector* algorithm. Shinghal and Toussaint [ST79b] divided a dictionary into portions containing words of the same length. A score was calculated for each word in the dictionary from the product of the transition probabilities between individual letters. This gives a probabilistic measure for the word based on a first order (bigram) Markov model. Words in each portion of the dictionary are then arranged in descending score order. Input words are first processed by the modified Viterbi algorithm (see section 6.6.2) to predict an output word. Although the modified Viterbi algorithm may produce a set of possible words, only the highest scoring one is used. The output word is then searched for in the appropriate length portion of the dictionary and the closest match, called a “mate”, is found. If the mate is an exact match then it is output as the correct word. If the match is not exact then a dictionary method is used which finds the scores of words in the neighbourhood of the mate. The word with the maximum score is selected as the correct word. The algorithm gives similar accuracy to a pure dictionary method but is approximately twice as fast.

Shingal [Shi83] extended this algorithm so that a number, $d \ge 1$, of the most likely candidate words output by the modified Viterbi algorithm are searched for in the dictionary, not just the highest scoring one. This variation also searches all the dictionary words of the desired length, rather than just those in the neighbourhood of the mate. Shinghal claimed this algorithm reduced the complexity of a dictionary search by three-quarters.

The Predictor-Corrector is known as a *hybrid-cascaded* algorithm because the bottom-up (modified Viterbi) and top-down (dictionary) methods are performed sequentially. Hull *et al.* [HSC83] proposed a *hybrid-integrated* algorithm which fully integrates the bottom-up and top-down processes.

The integrated algorithm, called the *Dictionary-Viterbi* algorithm, performs a Viterbi algorithm but at each stage of generating a word from the transition probabilities it verifies the generated sequence by checking that it occurs in the dictionary. The verification is facilitated by maintaining a vector of pointers into the trie structure in which the dictionary is stored. This allows the algorithm to rapidly check that the sequence of letters matches the start of at least one word in the dictionary. With bigram probabilities an 87% correction rate was achieved on a simulation of errors in machine-printed text, in comparison to 83% using the dictionary method alone and 39% with the Viterbi algorithm only [HSC83]. Trigram probabilities gave no improvement over the bigram probabilities when used in conjunction with the dictionary verification.

The Dictionary-Viterbi algorithm should have lower computation and storage needs than the Predictor-Corrector [SHC83]. Tests on simulated data suggest that it also gives slightly greater accuracy. This is because it integrates the bottom-up and top-down information, whereas the Predictor-Corrector simply uses the bottom-up transition probabilities to speed up the top-down dictionary search. The Predictor-Correct is therefore only as accurate as a pure dictionary search, while the Dictionary-Viterbi is slightly more accurate.

Sinha and Prasada [SP88] presented a hybrid method which, in addition to top-down and bottom-up information, also uses the probabilities that letters are corruptions of other letters. These probabilities are called *channel characteristics* or *confusion probabilities*. The algorithm starts by using the channel characteristics to substitute letters of the input word for the letters with which they are most frequently confused, e.g., ‘g’ and ‘q’ are confused with high frequency. This substitution process generates several alternative strings called *aliases*.

These aliases are then matched against a partial dictionary (a reduced set of the most frequently occurring words from a complete dictionary). Aliases which are found in the partial dictionary are graded according to the confusion probabilities which created them. If no aliases are found in the dictionary then the correct word is estimated using the modified Viterbi algorithm on the input word. The method has been applied to machine-printed character recognition in a two-pass system. The first pass finds aliases which are in the partial dictionary and modifies the confusion probabilities according to the substitutions in those aliases. This is intended to bias the second pass towards the dominant font in the document. They achieved 98% correct recognition on a variety of fonts.

Sinha *et al.* [SPH93] used a variant of this method which uses the modified Viterbi algorithm to find aliases. It also uses a string matching algorithm which finds words with common trigrams that are similar to the input word. This creates even more aliases. The collection of aliases are then costed and the lowest cost word is selected as the correct word. The complex cost is based on the transition probabilities, string edit distance, the letters' characteristics in relation to the expected characteristics for those letters (height and width, length of their chain codes) and confusion probabilities. Again, the initial classifications are used to bias the confusion probabilities towards the dominant font. Reliable initial classifications which form words are also used to create a transient dictionary which supplements the standard dictionary. The aim of this is to identify non-dictionary words which occur several times in the particular document. The method also uses some heuristics to handle words containing numerals and punctuation, and touching characters. However, these heuristics are simplistic and rely on high accuracy identification of the numerals and punctuation. The segmentation of touching characters is based solely on their combined width and so could not detect, for example, touching pairs of 'l's and 'i's, whose total width is often no greater than that of a 'w' or 'm'. The method generally relies on well separated characters. It is designed for use on good quality machine-printed text, and it is well suited to such documents. However, it is not robust enough for extension to handwritten text. The lower accuracy initial recognition would cause the heuristics to fail and segmentation problems would not be adequately resolved.

Channel characteristics were also used by Baird *et al.* [BKP86] [KPB87] in their experiments on words vectorized by Pavlidis's vectorization algorithm [Pav86] (see section 3.2). First they attempted to verify input words using the Unix 'spell' program. If an input word is rejected, then a list of variant spellings is generated, costed according to the confusion probabilities between letters of the variant word and letters of the input word. The variants are taken in ascending cost order and checked with 'spell'. If they are still invalid, further variants are generated from the rejected variant. The process continues until a valid variant spelling is found or the cost exceeds a limit and the word is rejected as unrecognizable. The method is slower than the previous hybrid methods but accuracy is high on good quality machine-printed text.

Although hybrid methods have achieved high accuracy on machine-printed text, the authors have not attempted to correct handprinted or cursive text. This is because segmentation is much more difficult on these types of text. The same levels of accuracy cannot be expected on handwritten words unless accurate segmentation can also be attained.

Recently, methods have been developed which use Markov models of the formation of characters. Characters are over-segmented into subcharacter segments. Whole characters are modeled as sequences of subcharacter transitions in a similar way to which words are modeled as sequences of character transitions in the above contextual methods. In some cases these character models have also been combined with contextual word models.

Bose and Kuo [BK94] used Pavlidis's vectorization (see section 3.2) and over-segmented words with vertical line segmentation. They then attempted to extract structural primitives from the vectors in each subcharacter segment. This approach suffers from the same problems as other structural methods (see sections 4.3.1 and 6.3), namely that too many errors occur in primitive extraction from noisy or degraded images. The method was reasonably accurate on a small test on machine-printed text with simulated blurring and overlap but would not be robust enough for handwritten, or even handprinted, text in a real environment. The greater levels of degradation, noise and style variation would severely hamper the primitive extraction and significantly complicate the Markov model of characters.

Chen *et al.* [CKZ94] also used a single pass vertical over-segmentation to produce subcharacter segments. They extracted moments, pixel distribution measurements, geometric and topological features from each segment and attempted to recognize them based on subcharacter transition probabilities. Each character was assumed to be segmented into no more than three subcharacters. The subcharacter segments were therefore recognized individually as being the left part, middle part or right part of one of the possible characters. The modified Viterbi algorithm is used to find the most likely selection of the candidate classifications, according to both a model of the subcharacter transitions and a Markov model of a small lexicon. Using a 30-nearest-neighbour classifier they achieved a 72.3% recognition rate by approximately matching the 20 best strings generated by the modified Viterbi algorithm to a 271-word lexicon. Although this result is one of the best published for cursive script recognition it is still too low for practical use. Also it is a very restricted experiment (only 94 words were tested) and the data comes from an unsegmented version of the CEDAR database (see section 2.6.2) and is much cleaner and less degraded than other databases such as NIST or the airline tickets used in chapter 7. The method cannot be expected to perform as well in larger tests using noisy data.

The problem with Chen *et al.*'s method is similar to that with Bose and Kuo's method. The individual subcharacters cannot be accurately identified in a noisy environment. They may actually be more difficult to recognize than the whole characters. There are three times as many classes of subcharacter as there are classes of character. Also, the segmentation will not always split characters into the same portions, so the amount of a character which is considered to be its left part (or alternatively middle or right part) varies for each case. In noisy and degraded images some subcharacter segments contain small blobs which represent a major part of the whole character but on their own could be part of almost any character. Alternatively the blobs could be trivial parts of a character (such as ligatures) which are relatively insignificant when viewed as part of the whole but seem much more important because they are in a segment on their own. Although the method aims to identify these blobs based on the surrounding subcharacters, this is very difficult when the surrounding segments may also contain indistinguishable blobs. It is much easier to recognize whole characters than these subparts.

Chapter 7 presents a similar over-segmentation method which recognizes whole characters in the first instance rather than trying to identify small subcharacters. It is suggested that the most effective strategy for segmentation is adaptive segmentation, as described in section 5.7, based on contextual correction from higher levels. An architecture for this contextual correction and adaptive segmentation is presented in chapter 8.

6.6.4. Higher Level Context

The application of higher level context (grammatical and semantic) to character recognition has been very limited. This is partly due to the difficulty of such a task and partly because there are still many improvements that can be made to the speed, accuracy and scope of word level lexical context.

Grammatical (or syntactic or linguistic) models of language are difficult to generate from examples. Complete models of major languages do not exist and even obtaining grammars for restricted subsets of languages is extremely difficult. The lack of available data from which such grammars can be derived is one of the main obstacles to grammatical context mechanisms. The field of computational lexicology (see Amsler [Ams82]) has arisen to address this problem but it is still in its infancy.

Similarly, data for determining semantic context is scarce. The use of machine-readable dictionaries with definitions is possible. Semantic consistency between words in a sentence can be found by matching words in their definitions [Eve93]. For example, the definitions of 'bank' and 'deposit' may both contain the word 'money'. This is a very limited approach, however, and is prone to making erroneous matches. Designing a semantic context checker for a general language is an extremely ambitious goal. Applications of semantic context to character recognition are currently limited to very small domains.

Srihari and Bozinovic [SB83] extended a hybrid contextual word recognizer to include a stage which selects the most grammatically appropriate word from a candidate set. Rejection or acceptance of words was determined by a string grammar, parsed by an augmented transition network (Woods [Woo70]). This was not sufficiently effective because the sentence level context did not take into account the confidence factors of words. A more complex grammar, which can cater for words with a degree of uncertainty, is required.

Stringa [Str80] presented a recognition system which included both grammatical and semantic context in a restricted application: Italian postal address recognition. Addresses generally conform to a small set of rules which can be encoded in a grammar, and have semantic connections between different portions of the address. For example, postcodes have a strict format, and elements of the post code are semantically related to the name of the city and name of the province. Stringa's system, called SARI (Sistema Automatico Riconoscimento Indirizzi), is only concerned with sorting mail with respect to city so only the postcode, city and province are considered. However, given a database of postal addresses, further contextual verification could be applied to other parts of the address block. SARI uses a feedforward method which computes similarity measures between the input and possible matches at each stage. No decision is made until the top level so that the loss of information, caused by decision making, is minimized. The method is accurate enough for practical use on machine-printed addresses, but it does not tackle the problem of segmentation. It relies on characters being sufficiently separated for easy segmentation. Application of the method to hand-printed or handwritten addresses would require considerable extensions of the basic approach to handle segmentation.

Little more work has been done to apply syntax and semantics to character recognition. However, the large field of linguistics (see Fromkin and Rodman [FR88] for a good introduction) has for many years been developing grammars and semantic models for other applications. It is likely that these will be used in contextual character recognition in the future.

6.7. Conclusions

Statistical classification has a well established mathematical basis in statistical theory. Statistical classification methods are well suited to character recognition because they are generally tolerant to noise and distortion. The methods are based on modelling the underlying class probability distributions. These are either modeled exactly, estimated or represented directly from training samples using one of the many techniques which apply to different circumstances. Unsupervised techniques are also applicable, though these are less common in character recognition. Modelling the distributions of training samples is sometimes slow but acceptable for practical use. The main difficulty in learning is that to accurately represent the distributions a large number of samples are required. Obtaining a sufficient quantity of samples is not always possible and compromises have to be made. Although using larger numbers of samples increases the training time, this is preferable because accuracy is the main concern. Once learning is complete, the operation of the classifiers, in most cases, no longer depends on the size of the training set. Statistical classifiers are usually much faster in operation than structural methods.

Neural classifiers are mostly replications of statistical classifiers and share the same advantages in speed of operation. However, they have become more popular because of their similarity to brain mechanisms and because they are much easier to use. Although many neural networks do make assumptions about the underlying class distributions, the users do not have to concern themselves with them. While some statisticians argue that this is unwise, it is very convenient to be able to simply present a neural learning mechanism with a set of training exemplars and have it learn them with high accuracy. Several neural classifiers are capable of forming arbitrarily complex decision boundaries, giving very high accuracy on self tests of the training set. This is sufficient to satisfy most pattern recognition researchers, even though the network's generalization to unseen data may not be as good as a properly designed statistical equivalent.

As the networks get more powerful, their statistical equivalents become more complicated. Neural networks provide more intuitive ways of understanding the training and operation of these classification models. Some, such as multilayer perceptrons, are extremely flexible representations which cover a range of parametric and nonparametric statistical models [Sar94]. A further advantage of some neural networks is that they can be trained on-line, i.e., they can learn from unstored examples while in use. Statistical classifiers require all exemplars to be stored so that they are always accessible during learning.

The main weakness of statistical and neural classifiers is that they rely on the extracted features to provide sufficient information to discriminate classes. However, many powerful features have been developed for character recognition (see section 2.5 and chapter 5) so the classifiers generally perform well in this application. The main limitations of features for statistical and neural character classification are that they do not represent structural relationships in the characters and character patterns cannot usually be generated from the features. The latter is not always true as some features can be used to reconstruct, or partially reconstruct, characters, e.g., Fourier descriptors and the RD/SA features of chapter 5. However, it is generally true that these features represent shape rather than structure. This limitation is addressed by structural classification.

Structural classification aims to classify characters based on extracted primitives and the relationships between them. The difficulty with this approach is that it relies on accurate extraction of the primitives. In practice, noise and distortion in the image cause errors in the extraction. Primitives may be incorrectly identified, which can be very detrimental as some techniques are highly sensitive to the type of primitive; the distinction between a straight line and a curve can be crucial. Primitives may also be missed altogether by the extraction process. Unwanted primitives may be created by noise or the idiosyncrasies of thinning and vectorization algorithms. Structural methods are generally very sensitive to noise and variation in character styles. Although the most complicated methods have achieved high accuracy on handwritten digits, they have yet to prove robust enough for the more difficult problem of upper and lower case letter recognition in a real (noisy) environment.

Structural character recognition has also been limited so far by its computational complexity. The size of template sets required to represent character classes makes structural matching too slow for practical applications. Learning of structural relationships is very complicated and also very slow. Statistical and neural classification techniques offer faster, more practical alternatives and are generally comparable or better in their accuracy. Structural methods have useful applications in contextual processing, however, where their implementation for higher level structural matching, such as dictionary checking, is simpler and more intuitive than statistical or neural alternatives.

Contextual processing of isolated characters is essential to remove ambiguity and correct errors. Most of these techniques use structural methods, mainly string matching and formal grammars, to check the spelling of input words. This is done either by a direct comparison to dictionary words or by comparison to a Markov model of legal transitions between letters. Most methods score legal candidate strings and select the highest scoring one as the correct word.

Dictionary matching methods are accurate but slow, whereas Markov methods are fast but less accurate. Hybrid methods attempt to combine the two approaches to gain the benefits of both. This is usually done by using one method to restrict the search space of the other. Some methods, such as the Dictionary-Viterbi algorithm [HSC83], use information from both models and can improve on the performance of a pure dictionary method. Most of these techniques have been applied only to machine-printed text. Contextual processing of handwritten text requires a consideration of segmentation.

As discussed in section 2.4, accurate segmentation requires that the process be integrated with classification. The segmentation which gives the highest confidence output is selected as the correct one. A logical extension of this is to further integrate the segmentation with the contextual mechanisms. Attempts to integrate segmentation in this manner have used Markov models of words combined with Markov models of characters (probabilistic models of the transitions between subcharacters). The problem with this approach is that subcharacters are more difficult to accurately recognize than whole characters, particularly in noisy environments. Chapter 7 will present an approach which performs multiple segmentations integrated with a recognition of whole characters and an approximate dictionary matching contextual mechanism. Chapter 8 extends the idea further to integrate higher level context with the adaptive segmentation approach described in section 5.7.

Higher level syntactic and semantic context has rarely been successfully applied to character recognition because of the difficulty of producing effective models of these complex areas of knowledge. A few methods have used such context in very limited domains. It is likely that this area will develop significantly in future research.

The following chapters present models for contextual processing of handprinted and handwritten text. Segmentation is a key issue, as is ensuring contextual sensibility over all the applicable knowledge domains.

6.8. Nomenclature

$\forall$	For all.
$\in$	Element of.
$\left[\begin{matrix} n \\ m \end{matrix} \right]$	Combinations of m elements chosen from n.
$\lvert \mathbf{M} \rvert$	Matrix determinant or vector modulus.
$\mathbf{M}^t$	Matrix (or vector) transpose.
$P(X)$	Probability of X.
$P(X \mid Y)$	Probability of X given Y.
$P(X,Y)$	Probability that Y is correct when X is observed.

α, β	Learning rates.
Δ	End of word.
μ_i	Mean vector of class *i*.
Γ_i	Covariance matrix for class *i*.
$(\mathbf{a}^{(p)}, \mathbf{b}^{(p)})$	Pattern pair *p*.
b	Margin.
C_i	Class *i*.
$d()$	Discriminant function.
D_{ij}	Binary digram for pairings of the *i*th and *j*th letter of a word.
$f()$	Activation function.
net	Weighted sum of inputs.
$o, o_{\max}, o_{\min}$	Output activation, maximum and minimum values.
$S()$	Sigmoid function.
T, T1, T2	Threshold values.
$\mathbf{w}$	Weight vector.
w_0	Threshold weight.
w_{ij}	Weight from neuron *i* to neuron *j*.
$\mathbf{x}$	Vector of input (feature) measurements.

Chapter 7

Integration of Segmentation and Dictionary Matching

7.1. Summary

In chapter 2, techniques for segmentation of words into individual characters were reviewed and it was concluded that segmentation must be combined with classification if the correct division is to be found. Casey and Nagy's original proposal [CN82] was for a recursive combination of segmentation and classification to try segmentation alternatives until a high confidence classification was found. In chapter 5, this idea was extended to an adaptive segmentation which searches for the correct separation of each character directed by hypothesized features for a hypothesized character class. In chapter 6, it was concluded that accurate classification of isolated characters requires an integration of the classification mechanism with contextual mechanisms.

This chapter and chapter 8 extend these concepts further. Chapter 8 will propose a hierarchical network architecture which integrates segmentation with both classification and contextual processing. This chapter explores the same idea but in a much simpler architecture.

Section 7.2 discusses approaches to the integration of segmentation with contextual processing. Section 7.3 describes a simple over-segmentation method used to break words into characters or parts of characters. A single character may be contained in one or more of these segments so they must be grouped together to form characters. Section 7.4 explains how the segment groups are then assembled into multiple possible sequences of characters which make up the whole word. Each possible sequence has its segment groups classified; a small candidate set of characters is produced for each group. Section 7.5 describes a method for approximately matching these sequences of character candidates against a dictionary and scoring matched words to determine their relative certainty. Experimental results are recorded in section 7.6. Section 7.7 discusses a method for choosing a good initial segmentation for use in an alternative adaptive segmentation. Section 7.8 draws conclusions about the approach and section 7.9 describes the notation and symbols used in this chapter.

7.2. Integrated Approaches to Context Sensitive Segmentation

Segmentation of characters cannot be accurately performed in a single pass and must be integrated with the classification stages of character recognition. In section 2.4, Casey and Nagy's proposal of recursive segmentation and classification was discussed [CN82]. This involves performing an initial segmentation, classifying the segments into character classes with a measure of confidence, and re-segmenting and reclassifying if the confidence is too low. One of the conclusions of chapter 6 was that segmentation should be further integrated with contextual processing to obtain more accurate results.

Existing approaches to the integration of segmentation and contextual processing (Bose and Kuo [BK94], Chen *et al.* [CKZ94]) have used Markov models of word level context (see section 6.6.2) combined with similar Markov models of over-segmented characters (see section 6.6.3).

One of the problems with single-pass segmentation is that algorithms which attempt to segment words precisely into characters often fail to detect all the cases where characters are merged together. One approach to ensuring that all the correct divisions of characters are found is to over-segment the words. Rather than attempting to segment a word precisely into its component characters, over-segmentation uses more segments than are required. Some characters will be divided into more than one segment, but it is hoped that all the merged characters will be separated in the process. Over-segmentation does not guarantee that all joins between merged characters will be detected but it does greatly increase the incidence of finding them.

Having over-segmented the words, Bose and Kuo's and Chen *et al.*'s approaches to context integration attempt to recognize each segment individually. Since characters may lie across more than one segment, the contents of individual segments are treated as sub-parts of characters (*subcharacters*). The probabilistic transitions between possible subcharacters are modeled as a Markov process and this is used as the basis for recognition of whole characters. Sequences of non-overlapping groupings of subcharacters into whole characters, which use all the available segments, are called *character segmentations*. The goal of the integrated methods is to find the character segmentation whose character classifications form the highest confidence word.

The problem with this approach is that it is very difficult to accurately identify the subcharacters for reasons discussed in section 6.6.3. Briefly, there are four main reasons why subcharacters are harder to identify than whole characters. Firstly (at least in Chen *et al.*'s method) there are many more classes for subcharacters than whole characters so much greater discriminatory detail is needed to distinguish them. Secondly, subcharacters are smaller and contain less discriminatory information; many subcharacter blobs could be part of almost any character. Thirdly, the initial over-segmentation of characters cannot always divide characters of the same class into the same sub-parts each time so many different models of characters are needed to represent every possibility. Finally, trivial sub-parts of characters may be given too much significance because they are in a segment on their own.

Another problem with the existing approaches is that Markov models of legal character transitions are less accurate than dictionary matching methods of contextual correction. A pure dictionary method or hybrid method would be more effective (see section 6.3.3).

One other approach to the recognition of over-segmented words was used by Bozinovic and Srihari [BS89]. Rather than attempting to identify the individual segments as subcharacters, they grouped the subcharacter segments together and attempted to recognize them as whole characters. This made the isolated character level recognition more accurate. They also used a dictionary matching method to perform the contextual checking of words.

Bozinovic and Srihari tested their method on cursive script. Although their results are among the best reported for off-line cursive script, their test data was of very high quality. Their best result of 78% was on words deliberately written with no slant, and although the letters were joined to each other by connecting strokes there was virtually no horizontal overlap between characters. They also required that the lower contour of each word be continuous. This meant that using local minima of the lower contour as a guide, it was very easy to find a set of vertical divisions which contained all the desired segmentation lines for the word. Given the ease of segmentation and the quality of the characters, this data can be considered comparable to handprinted text. It should also be noted that the 78% result was obtained after a retraining phase which relearned the parameters of the isolated character recognition based on an initial recognition of the test data. The best results without this retraining were 77% using a 710-word dictionary and 48% with a 7800-word dictionary.

Bozinovic and Srihari's dictionary matching algorithm starts at the beginning of the word and looks for the highest certainty character classification from the first segment, the first segment pair and the first segment triple. The highest certainty character is selected as the first character in the word and the algorithm advances the length of the corresponding segment group, i.e., if the highest certainty character was found in the classification of the segment pair then the algorithm advances two segments. The method then repeats from the new starting position to determine the second character and so on. At each stage the contextual validity of the partial word is determined by matching the prefix against a trie-structured dictionary. If the prefix does not exist in the trie then it is rejected and the algorithm backtracks to a less certain alternative.

Not every possible character segmentation is checked so the method is biased by the ordering of the search. Although some backtracking is allowed, the method will tend to find words which fit the high certainty character classifications amongst their first few letters, even if these classifications are incorrect. Another problem is that since the algorithm only classifies segment groups which start at the end of previously accepted groups, some segment groups may be missed out. Many high certainty characters will therefore not be found because they are in the unclassified groups.

A better method would allow high certainty character classifications to be found regardless of where they occur in the word. It would also use these high certainty characters as the basis for the dictionary matching, rather than always basing it on the first letters of the word. This is the approach taken by the adaptive segmentation approach, described in section 5.7 and intended to be used in the contextual system of chapter 8. The simpler method presented in this chapter achieves these objectives by considering every possible character segmentation.

On one hand this chapter presents an alternative method for integrating segmentation with word level context checking. On the other, it is a step towards building an independent word level contextual module for use in the larger system proposed in chapter 8. A scoring system is developed for assigning classification certainties to matched words. The matching and scoring methods can be used to form a word level context module which classifies sequences of character candidate outputs and can be used in the proposed hierarchical network. This chapter also presents a simple over-segmentation system for handprinted text and an algorithm for constructing legal character segmentations.

7.3. Over-Segmentation

Firstly, it is assumed that we are dealing with individual words. This is not an unreasonable assumption because existing methods for identifying text regions and dividing them into lines of text are sufficiently accurate at performing this task. Although words on a text line will sometimes touch, most words can be segmented by an analysis of the gaps between areas of black pixels. A totally unconstrained recognizer will require a more advanced method of text line segmentation, probably involving an integrated over-segmentation and contextual recognition of sentences, similar to this chapter's approach to word segmentation. This chapter will assume that a gap-based segmentation into words has been used and the input to the method is a representation of a word image. The particular representation used in this research is the outline format, described in section 2.2.2.4, which was also used in chapter 5.

The initial over-segmentation used in this method is a single-pass operation. An adaptive system would have to select an initial sequence from this over-segmentation which represents a good first guess at the true character segmentation. This selection is avoided for the moment by considering all the possible character segmentations which can be made from the over-segmentation. A method for choosing a good initial segmentation is discussed in section 7.7.

The aim of over-segmentation is to find a compromise between too much and too little segmentation. The method must find all the correct divisions of the word and as few as possible incorrect ones. Most methods which use over-segmentation assume that after the process each character will lie in not more than a fixed number of segments. Three is the usual number. To simplify the tests in this chapter it is assumed that each character is contained in at most two segments; however, the integrated recognition method can easily be modified for the case where they are contained in three.

Common methods for segmenting use vertical lines to divide the word. This is reasonably effective on machine-printed text and most handprinted text. For cursive script a more complicated division is required as much greater overlap occurs. Section 5.7 proposed a more powerful cursive word segmentation. This chapter will limit itself to handprinted text where only vertical segmentations are required. This greatly simplifies the construction of character segmentations because it can be safely assumed that segments can only be joined to their immediate left or right neighbours. Non-vertical segmentation introduces another dimension and the number of possible character segmentations increases accordingly.

Segmentation methods have had most success on machine-printed text. These methods detect possible segmentation points based on pixel and profile projections. These are vertical histograms of either the number of black pixels in a column (pixel projection) or the lowest and/or highest black pixel in a column (profile projection). The pixel projection method assumes that columns where characters are lightly touching or separated will contain few black pixels, whereas columns in the middle of characters will contain many, e.g., Kahan *et al.* [BKP86] KPB87]. This works on machine-printed text which is lightly touching. For handprinted text where more overlap can occur the profile projection is a better approach. Possible segmentation points are detected at local minima and maxima of the upper and lower profile projections (or alternatively the upper and lower edges of the word outlines), e.g., Liang *et al.* [LSA94].

While these methods focus on splitting merged characters, the method used in this chapter is more concerned with broken characters. The method was intended to be used in a system to automatically read airline tickets. These tickets are roughly 75% machine-printed and 25% handprinted. Examination of a sample of handprinted tickets showed that the main segmentation problem with them was broken characters. Letters rarely touched and the low quality of the data caused many characters to fragment.

The over-segmentation method tested here allows characters to vertically overlap their neighbours provided they do not merge together. It assumes that there are no merged characters. This is an oversimplification of the problem which is reasonably effective for the airline ticket data but not for general use. The reason for this oversimplification is that it allows the method to make the further assumption that whole characters will be contained in at most two segments. Testing of the approach is therefore much easier and much less computationally expensive on a serial machine.

A full segmentation system for handprinted characters will require segmentation of merged characters, probably using profile projections. Algorithms for this have been proposed by several researchers, e.g., Shridhar and Badrelin [SB86] [SB87], Kimura and Shridhar [KS91] [KS92], Bozinovic and Srihari [BS89], and Leedham and Friday [LF] (see section 2.4). Bozinovic and Srihari then assume that characters are contained in at most three segments. This seems to be a reasonable number of segments for over-segmentation performed by these algorithms. The method presented in this chapter can be extended to a full system with the assumption of three-segment characters. However, it was decided that the simplified version of the problem was sufficient for testing the basic approach.

The method for over-segmentation of non-merged handprinted or machine-printed characters works from the outlines of words. Each connected component of the word is represented by a single outer level outline loop, possibly with inner loops (see section 2.2.2.4). These loops are arranged in left to right order, based on the x coordinate of their leftmost point. Vertical segmentations are performed at the left-hand edge of outer level loops. The placings of segmentation points are chosen according to the horizontal distance from their left-hand edge to the previous loop's right-hand edge. This distance must exceed a threshold for a segmentation to be performed.

Two different thresholds are used, LOOP_GAP1 and LOOP_GAP2. When an image is badly broken up into small loops it is preferable to use a large gap in order to prevent the creation of too many small segments. For larger loops it is safe to use smaller (actually negative) gaps to allow some horizontal overlap of characters. The area of the box containing each loop is compared to a constant, DOT_SIZE. Areas above DOT_SIZE use LOOP_GAP1 and those smaller or equal to DOT_SIZE use the smaller threshold, LOOP_GAP2.

Furthermore, the gap between the left-hand edge and the previous segmentation position must also exceed a threshold so as to avoid making divisions too close together. Again, two different thresholds, SEG_GAP1 and SEG_GAP2, are used according to the size of the loop.

Pseudo C code for the assignment of new segments is as follows:

```
set leftmost_pt equal to a large positive number
set rightmost_pt and prev_seg_posn equal to large negative numbers
while there are more outlines to read
{
 read the next outline loop into outline
 set level equal to the level number of outline
 set left and right equal to the leftmost and rightmost x coordinate of outline respectively
 set height and width equal to the height and width of outline respectively

 if (level == 0 &&                         /* if it is an outer level loop and */
    (left < leftmost_pt ||                 /* (we have started a new line or */
     (height * width > DOT_AREA &&         /* (it's a 'large' loop and */
      left - rightmost_pt >= LOOP_GAP1 &&  /* it's not too close to previous loop and */
      left - prev_seg_posn >= SEG_GAP1) || /* not too close to previous segmentation) or */
     (height * width <= DOT_AREA &&        /* (it's a 'small' loop and */
      left - rightmost_pt >= LOOP_GAP2 &&  /* it's not too close to previous loop and */
      left - prev_seg_posn >= SEG_GAP2)))  /* not too close to previous segmentation)) */
 {
   start a new segment and add outline to it
   leftmost_pt = left;                     /* reset leftmost point of previous loop */
   prev_seg_posn = left;                   /* set position of previous segmentation */
   if (right > rgtmost_pt)                 /* if the loop extends past rightmost_pt */
     rightmost_pt = right;                 /* reset rightmost_pt */
 }
 else                                      /* do not segment at this point */
 {
   add outline to the current segment
   if (level == 0)                         /* if it is an outer level loop*/
     leftmost_pt = left;                   /* reset leftmost point of previous loop */
   if (level == 0 &&                       /* if it's an outer level loop and it */
       right > rgtmost_pt)                 /* extends past rightmost_pt */
     rightmost_pt = right;                 /* reset rightmost_pt */
 }
}
```

All that is required now is to determine suitable values for DOT_SIZE, LOOP_GAP1, LOOP_GAP2, SEG_GAP1 and SEG_GAP2 which separate whole characters but avoid separating broken parts of characters if possible. For the airline tickets a range of values were tested and the assignments DOT_SIZE = 8, LOOP_GAP1 = -3, LOOP_GAP2 = 1, SEG_GAP1 = 5 and SEG_GAP2 = 3 were chosen. This was found to be the most effective, allowing for the separation of most overlapping (but not merged) whole characters with minimal separation of broken characters. Note that these values apply to characters scanned at 300 pixels per inch. Higher resolutions (or significantly larger or smaller characters at the same resolution) will require re-evaluation of these assignments.

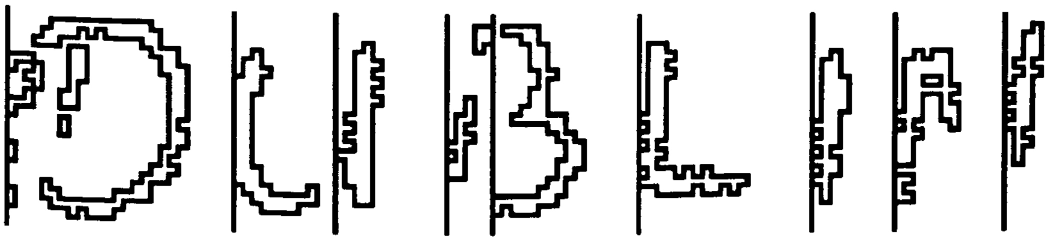
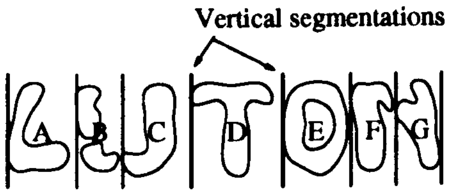


Figure 7.1 Example of the over-segmentation method for broken handprinted characters.

Figure 7.1 gives an example of the over-segmentation of an airport name, "DUBLIN," scanned from a real ticket. The vertical lines indicate the divisions of the word made by the algorithm. Some characters fall into two segments while others are contained in one. Note that not every component in the image causes a segmentation. The two outlines in the middle of the 'D', for example. These are both too far left of the previous rightmost point (the rightmost point of the curved section of the 'D'). Also the curved section of the 'D' and the small section at the top of the 'B' would create segmentations too close to the previous ones.

7.4. Constructing Legal Character Segmentations

Once the segmentation has been performed, the next task is to group the segments into potential characters. Since it is assumed that a character is contained in no more than two segments, this means constructing groups of either single segments or segment pairs. Each segment grouping is represented by a structure which stores all the relevant information (mainly pointers to the individual segments, the feature vector for the group, the candidate character classifications and their associated certainty measures). The ordering of segment groupings is chosen to aid the method for producing legal (non-overlapping) character segmentations.



Ordering: A - AB - B - BC - C - CD -
D - DE - E - EF - F - FG - G

Figure 7.2 Example of segment group ordering. Segment groups are either single segments (e.g., A) or pairs (e.g., AB). The correct character segmentation is A - BC - D - E - FG. However, many other incorrect, but legal, segmentations are possible.

The ordering of segment groupings is a simple list. Let *A* be the current segment and *B* be the next segment if it exists. Initially, *A* is the first single segment. The ordering starts with segment *A*. Then, while there are more single segments, the segment pair formed by *A* and *B* is added to the list followed by the single segment *B*, and then *A* is advanced to *B* and *B* is advanced to the next single segment. Figure 7.2 illustrates this ordering for a simple example.

Next, features are extracted from each segment group (treating segment pairs as if they were large single segments) and stored in the segment group structure. The features are classified by an isolated character recognizer and a candidate set of possible characters, and their associated certainties, are stored for each segment group.

In the experiments in this chapter, the features extracted are the RD/SA features of chapter 5 and the isolated character recognizer is a radial basis function (RBF) network trained by learning vector quantization (LVQ) (see section 6.4.4). This is essentially the same as the LVQ classifier used in chapter 5 but the distance to the codebook vectors is passed through a radial basis activation function which produces a maximum value at zero and approaches zero as it approaches infinity. This inversion of the distance is used as a certainty measure for the class represented by the codebook vector. The certainty measures are normalized over all the classes so that the total certainty is 1.0. The certainty is then treated as a pseudo probability. The RBF classifier therefore outputs a candidate set of the most likely characters along with their associated pseudo probabilities. The particular set of codebook vectors used for recognizing the upper case letters in these experiments is the one produced by training on the NIST Upper A data set, which gave 89.81% correct recognition on the previously unseen NIST Upper B test set using LVQ classification (see table 5.7).

The next step is to form the legal character segmentations. For s single segments there are $\mathit{fib}(s)$ legal character segmentations, where $\mathit{fib}(s)$ is the sth element of the Fibonacci series (1,2,3,5,8,13,...), e.g., $\mathit{fib}(5) = 8$. (Note that some people may include additional elements at the start of the series, e.g., 0,1,1,2,3,5,8,..., but for ease of implementation we shall assume the ordering given previously.) All of these will be generated and tested to determine which forms the most probable word.

The legal character segmentations are stored in a $\mathit{fib}(s) \times s$ array of pointers to segment group structures, where s is the number of single segments making up the word. The legal sequences are constructed by adding segment groups to the $\mathit{fib}(s)$ sequences which should contain them. The ordering of the constructed sequences is based on adding single segment groups or their immediately following pair groups in *Fibonacci ratio* between consecutive rows of partial sequences which end at the same point in the segmentation. For $\mathit{fib}(k)$ rows, the Fibonacci ratio is defined to be $\mathit{fib}(k-1){:}\mathit{fib}(k-2)$.

Let *total_groups* be the total number of segment group structures for the word and let these structures be numbered from 0 to $\mathit{total\_groups}-1$. Let p be the number of the current structure, which is initially 0 and advances in steps of two so it is always an odd number. Then while there are at least two group structures left (p and $p+1$) we add, in Fibonacci ratio, either the current or next structure to the end of any partial sequence which currently ends in structure $p-3$. We do the same for any partial sequence currently ending in structure $p-2$. In cases where there is only one group structure left ($p = \mathit{total\_groups}-1$) we add it to any partial sequence currently ending in structure $p-3$ or $p-2$.

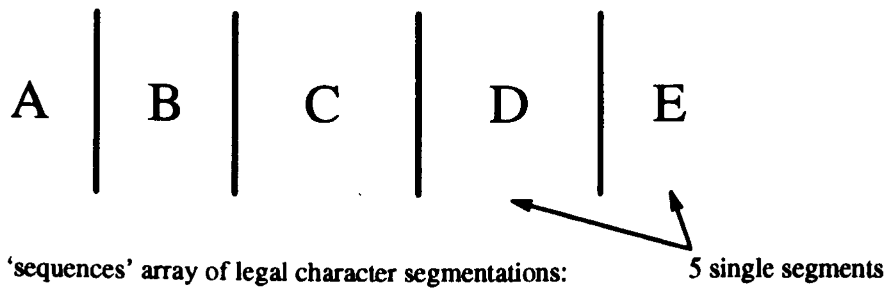
The following pseudo C code gives the full algorithm:

```
set s equal to the number of single segments making up the word
set total_groups equal to the total number of segment group structures
set group to the start of the linked list of segment groups

int no_seqs = fib(s);                          /* number of legal sequences */
struct segment_group *sequence[no_seqs][s];    /* array of legal sequences */
int *position;                                 /* index of next available position in a sequence */
int *last_group;                               /* last segment group added to a sequence */
int p, q;                                      /* group counter, sequence counter */
int set, set_size;                             /* size of set indicator, actual size of set */
int ratio, count,                              /* Fibonacci ratio cutoff, set counter */

allocate memory for position and last_group so that each array has no_seqs elements
for (q = 0; q < no_seqs; q++)                  /* for each sequence */
{
 for (p = 0; p < s; p++)                       /* initialize sequence array */
   sequence[q][p] = NULL;
 position[q] = 0;                              /* sequences start at index 0 */
 last_group[q] = -2;                           /* ensures groups will be added when q == 0 */
}
set = s;                                       /* fib(set) is the size of the set of rows */
                                               /* with the same last group */
for (p = 0; p < total_groups;)                 /* step through each segment group */
{
 if (group->next)                              /* there are at least two more groups to be added*/
 {
   set_size = fib(set);                        /* total number of sequences in each set */
   ratio = fib(set-1);                         /* cut off point for first group */
   count = 0;                                  /* start counting through the set */
   for (q = 0; q < no_seqs; q++)               /* step through all sequences */
   {
    if (last_group[q] == p-3 ||                /* if last group was p-3 or p-2 */
        last_group[q] == p-2)
    {
      if (count % set_size < ratio)              /* if before the cutoff point */
      {                                          /* add group p at next position */
        sequence[q][position[q]++] = group;      /* in sequence q */
        last_group[q] = p;                       /* record last group added */
      }
      else                                       /* if after the cutoff point */
      {                                          /* add group p+1 at next position */
        sequence[q][position[q]++] = group->next; /* in sequence q */
        last_group[q] = p+1;                     /* record last group added */
      }
      count++;                                   /* advance set counter */
    }
  }
  set--;                                         /* size of set decreases each time */
  group = group->next->next;                     /* advance to group p+2 */
  p += 2;                                        /* advance p */
 }
 else                                            /* only one segment group remains */
 {
  for (q = 0; q < no_seqs; q++)                  /* step through all sequences */
    if (last_group[q] == p-3 ||                  /* if last group was p-3 or p-2 */
        last_group[q] == p-2)
    {                                            /* add group p at next position */
      sequence[q][position[q]++] = group;        /* in sequence q */
      last_group[q] = p;                         /* record last group added */
    }
  group = group->next;                           /* this should advance to NULL */
  p++;                                           /* advance p */
 }
}
```

The ordering of legal character segmentations by this algorithm is illustrated in figure 7.3. Notice how single segments and segment pairs are distributed in Fibonacci ratio after their preceding groups. For example, after the set of partial sequences ending at group ‘A’ (sequences 0 to 4) there are three instances of group 'B' and two of group 'BC'. The ratio 3:2 is a Fibonacci ratio.



'sequences' array of legal character segmentations:

	0	1	2	3	4
0	A	B	C	D	E
1	A	B	C	DE	-
2	A	B	CD	E	-
3	A	BC	D	E	-
4	A	BC	DE	-	-
5	AB	C	D	E	-
6	AB	C	DE	-	-
7	AB	CD	E	-	-

8 legal segmentations (fib(5) = 8)

Figure 7.3 Example of legal character segmentation ordering in the sequences array. In this case there are five single segments and therefore eight legal sequences of segment groupings. The letters in the array indicate the segment grouping pointed to by that element. '-' indicates a null segment grouping.

Legal character segmentations can now be extracted from the array and passed to the context checking procedure.

These methods can easily be extended to handle the ordering and sequencing of three-segment groups. This will be necessary if the over-segmentation method is developed for segmenting merged characters.

7.5. Approximate Dictionary Matching

Initial attempts at approximate string matching used a program called *agrep*, written by Sun Wu and Udi Manber. It uses their own approximate string matching algorithm [WM91] developed from the exact string matching method of Baeza-Yates and Gonnet [BG89].

Agrep is similar to the *grep* family of string matching programs found on UNIX, but it allows matching of arbitrary regular expressions with insertion, deletion and substitution errors. This is extremely powerful and fast for searching large text files for approximately matched patterns. The use of regular expressions allows candidate sets of characters to be matched in a particular position, e.g.,

```
agrep '^[WXYZ][STUV][MNOPQR][IJKL][CDEFG][ABH]$' airports
```

would exactly match any six-letter word in the file called "airports" with 'W', 'X', 'Y' or 'Z' in its first position, 'S', 'T', 'U' or 'V' in its second position, and so on, e.g., "ZURICH".

Agrep is also capable of counting the total number of insertion, deletion and substitution errors occurring in an approximate match and outputing the best match found, i.e., the match with the least errors. This facility was used for the initial experiments on airport names extracted from scans of real airline tickets.

A pattern string was constructed for each legal character segmentation by taking a set of candidate characters for each of its segment groups, enclosing them in square brackets and concatenating them to form a string similar to that in the above example. The '^' (start of line) and '$' (end of line) symbols are also added to constrain the pattern to completely match whole lines of the dictionary. The dictionary contains one word per line so this means that patterns must match whole words.

The set of candidate characters is selected so as to contain all potential character classifications (produced by the RBF classifier described in section 7.4) whose certainty is above a threshold value. This use of candidate sets reduces the search space of the match. However, the certainties of characters which are left out of a candidate set are retained so that they may be used later when scoring words. If no characters have certainties above the threshold then the segment group is treated as a "wild card" — all characters are allowed in the candidate set so no errors occur when it is matched.

Agrep is called for each character segmentation and the best match is found in each case. The best matches are each given a score according to the certainties of their characters in the matched positions within their character segmentations. Often the same word is matched more than once. In this case only the highest score for that word is stored. The words are sorted into score order and are outputed, together with their scores. The highest scoring words can be extracted as a candidate set for input to further contextual processes, or the highest scoring word can be selected as the final classification.

It soon became obvious that there were problems with this initial approach. Because agrep allows deletions and insertions of characters, many words were matched which did not precisely fit the segmentation. Deletions and insertions are very useful if the segmentation is incorrect as it can match words when more than one character is contained in a segment group, or when one character is spread across more than one segment group. However, since we are checking all possible segmentations we can reasonably expect that one of these segmentations is the correct one. To allow matching with deletions and insertions is therefore counterproductive. Matching of words which do not precisely fit a segmentation is undesirable as it causes matching of too many words.

The output below shows the set of 44 candidate airport names matched using agrep for the word "DUBLIN" shown in figure 7.1. The score next to each word is the average character certainty for the matched characters in the word.

DUBLIN	15.938	SANJUAN	6.674	STANSTED	4.109
HELSINKI	15.260	LINCOLN	6.484	BARBADOS	4.004
ZURICH	10.524	STLUCIA	6.279	BORDEAUX	3.869
GLASGOW	9.259	GATWICK	6.103	VERONA	3.817
AUCKLAND	8.720	KENNEDY	5.889	VIENNA	3.801
NAPLES	8.421	GENEVA	5.804	GENOA	3.720
ANTIGUA	8.396	MOSCOW	5.413	MARSEILLE	3.639
TOULOUSE	8.309	SIDNEY	5.353	AMSTERDAM	3.589
LUTON	8.190	PRAGUE	5.041	VANCOUVER	3.526
BALTIMORE	7.403	MALAGA	5.010	CORFU	3.515
ATLANTA	7.263	BERMUDA	4.828	MADRID	3.459
BRUSSELS	7.067	KINGSTON	4.754	MALTA	2.402
TENERIFE	6.920	EDMONTON	4.662	PARIS	2.344
EDINBURGH	6.851	HEATHROW	4.296	STOCKHOLM	1.397
FRANKFURT	6.768	NANTES	4.129		

Although in this case the highest scoring word is the correct one, it is often desirable to output a candidate set for further contextual processing as the correct word will sometimes come a close second or third. The size of this candidate set is clearly far too great. It contains almost the entire dictionary (which in this case consisted of 55 airport names). The other tested examples produced similar sized word sets.

Examination of the pattern strings and matched words showed that a large number of the incorrect words were being matched with insertions and deletions, while the match of the correct word contained only substitution errors or none at all. The next logical step is therefore to disallow insertion and deletion errors in the approximate match.

Unfortunately, this is not easy to accomplish using agrep. Agrep does not have a facility for completely eliminating particular types of errors. Instead one can individually set the cost of these operations so that, for example, deletions and insertions cost 3 while substitutions only cost 1. However, the way in which agrep handles costed errors limits the allowable cost values to small integers. One cannot therefore set deletions and insertions to a very high cost so they are never accepted. One has to use lower values which means that words containing these errors can sometimes still have a lower cost than a word with several substitution errors.

It is possible to reduce the incidence of accepted insertions and deletions by rejecting any matched word which is not the same length as the character segmentation. However, by this time the advantages of using agrep are minimal. Little of the functionality of agrep is actually being used and the system is inefficient as it is having to perform matching with insertion and deletion errors and then having to try to reject them later. Also, from a practical point of view, the agrep code proved to be very difficult to integrate into the rest of the system as it is almost entirely free of comments.

At this point in the research it was decided to switch to a simpler system of checking the sequence of candidate character sets against dictionary words of the same length and counting the number of substitution errors. Any words which have a number of substitution errors less than or equal to half the length of the character segmentation are accepted as word candidates.

The new output set, shown below, is under half the size using this new method. There are still a large number of candidates though. Also the correct word, "DUBLIN," has slipped down to second place.

HELSINKI	15.593	PRAGUE	5.930	ANTIGUA	5.167
DUBLIN	14.945	NEWQUAY	5.822	BELFAST	5.008
GLASGOW	9.675	SIDNEY	5.561	MALAGA	4.838
LINCOLN	9.146	GATWICK	5.491	EDMONTON	4.724
TENERIFE	7.450	PLYMOUTH	5.359	KENNEDY	4.674
SANJUAN	6.436	MADRID	5.346	AUCKLAND	4.673
MOSCOW	5.962	STLUCIA	5.286	NAPLES	3.755

Development of the method now has two goals. One is to further reduce the number of matched words so that only very closely matching words are considered. The other is to improve the accuracy of the method.

Accuracy can be improved by altering the way in which words are scored. Several variations have been tried. In the above experiments the score was determined by summing the certainties of the characters which make up the word and dividing by *n*, where *n* is the number of characters in the word. An alternative is to multiply the certainties. Using the product as it is or dividing it by *n* will heavily bias the score towards short words. To obtain a consistent score, the *n*th root of the product is taken.

Another strategy for improving the scoring is to divide the initial score by certain values. The values tested were:

1) the total number of substitution errors between the pattern string and the matched word;
2) the absolute difference between the length of the word and the number of single segments in the over-segmented word;
3) the average absolute difference between the segment group width and the average segment group width.

The first biases the results towards words with few errors. The second assumes the over-segmentation is close to the correct segmentation and so biases the results towards words which have few characters in segment pairs. These two strategies proved to be undesirable because of their assumption that the segmentation or classification is initially correct. Although this is often the case, the purpose of the integrated algorithm is to find the correct segmentation and classification in all cases, particularly those where the initial decisions are incorrect.

The third strategy is based on the assumption that the correct character segmentation will have segment groups which are of approximately the same width. We therefore divide by the average absolute difference from the average width. Other values considered were the total absolute difference from the average width across all the segment groups, and the maximum absolute difference. These were found to be less effective than the average absolute difference. The total absolute difference biases the results towards shorter words, while the maximum absolute difference places too much emphasis on one segment group. Dividing by the average absolute difference proved to be the most effective scoring system.

The number of candidate words can be reduced by increasing the certainty threshold of characters which are allowed in the character candidate sets. Limiting the number of character candidates will limit the number of word candidates. This can be detrimental to accuracy, however, because correct letters may be left out of the character candidate sets.

7.6. Results

The data used for testing was a set of 100 handprinted airport names. This data was initially obtained from real airline tickets; however, the number of handprinted ticket scans available was limited so further data was collected. The additional test data came from six writers, using either biros or fine felt pens, who were instructed to write between ten and twenty airport names which they selected arbitrarily from a dictionary of 112 airports.

The ticket data was converted to a bi-tonal representation from a greyscale scan. The additional data was binarized directly by the scanner. Both scans were at a resolution of 300 pixels per inch. The word images were stored as TIFF files and then converted to outline format (see section 2.2.2.4).

The greylevel thresholding used in the binarization of these images was a simple global division between black and white (see section 2.2.2.2) which left a large amount of background noise that would not occur with more sophisticated thresholding. It was desirable in this case to perform a preprocessing filling operation on the images to remove the many small spots of background noise. The minimal preprocessing approach (see chapter 4) would ideally avoid the need for this through the use of a better thresholding algorithm. The filling used was a very simple removal of outline loops which represented single pixels.

Although the simple segmentation method is not intended to separate merged characters, some mergings were allowed in the test data. The integrated method was able to cope with small numbers of unseparable characters, but fair testing of the method required words with many merged letters to be removed for the test set. Nineteen of the words collected from the tickets and additional samples were removed for this reason. Recognition of these words will require a more complex over-segmentation and an extension of the character segmentation construction method to allow three-segment characters (see section 7.3 and 7.4).

Three dictionaries were used. One contained 112 airport names including all the words in the test set. The second was a small dictionary of 710 common words randomly chosen from Roger Mitton's CUV2 computer-usable dictionary. This dictionary is based on the Oxford Advanced Learner's Dictionary of Current English and includes information on the frequency of words which allowed the more common words to be selected. The 710 words also include the 112 airports names. The third dictionary consisted of 7827 words randomly selected from the CUV2 dictionary with no regard to frequency, and including the 112 airport names.

Table 7.1 shows the results obtained in tests using the different scoring strategies, described in section 7.5. The first column indicates which scoring strategy is used. It is either none, 1 (dividing the basic score by the number of substitution errors in the match), 2 (dividing by the difference between the length of the word and the number of single segments in the over-segmented word), 1 & 2 (dividing by both values) or 3 (dividing by the average absolute difference between the segment group width and the average segment group width). For strategies 1 and 2, note that to avoid dividing by zero, and to distinguish between words with one error (or one difference in length) and those with none, we actually divide by the value plus one. The dictionary column indicates the size of dictionary used (112, 710 or 7827 words). The final two columns show the number of correct word classifications out of the

Scoring Strategy	Dictionary Size	Correct Words (%) with Basic Score:	
		Mean Certainty	Root of Product
none	112	77	91
none	710	51	76
none	7827	29	56
1	112	76	86
1	710	58	62
1	7827	20	36
2	112	74	85
2	710	56	69
2	7827	34	39
1 & 2	112	86	93
1 & 2	710	73	76
1 & 2	7827	53	56
3	112	91	95
3	710	78	85
3	7827	61	77

Table 7.1 Comparison of the basic scoring methods and scoring strategies.

100 test samples (or equivalently the percentage of correct word classifications) for experiments using different basic scores. The two types of basic score are the mean certainty of characters in the word, and the *n*th root of the product of the *n* characters in the word (see section 7.5). The certainty threshold for including characters in the character candidate sets is set at 0.075 in all of these tests.

Clearly the 'root of product' basic score gives much better results than the 'mean certainty'. The strategies of dividing by the number of substitution errors (plus one) and the difference in length from the expected segmentation (plus one) are individually detrimental when using the smallest dictionary but together give slightly greater accuracy than the basic score alone. On the 710-word dictionary the strategies are both effective using the 'mean certainty' basic score, but do not improve on the 'root of product' basic score. Similarly, on the 7827-word dictionary, the combined strategies improve on the 'mean certainty' but not on the 'root of product'.

As noted in section 7.5, these two strategies work well in cases where the initial over-segmentation and classification are close to perfect, but are biased against cases where they are not. This is undesirable as we want the scoring to work in all cases, particularly those where errors have been made. Also, the overall result seems to balance out and no improvement can be made on the 'root of product' basic score.

The third strategy was much better and gave considerable improvement on the larger dictionaries. Clearly, biasing the scoring towards words whose character segments have approximately the same width is a very effective strategy.

If we consider Bozinovic and Srihari's high quality cursive word data [BS89] to be comparable to this handprinted data, as suggested in section 7.2, then this method has significantly improved on their results, at least for words with no merged characters. It is interesting to note that whereas this method does not handle words with merged characters, Bozinovic and Srihari's method does not handle words with non-merged characters. Their best results, without retraining on the test data, were 77% correct on a 710-word dictionary and 48% correct on a 7800-word dictionary. Note that the dictionaries used in this chapter were chosen to be comparable in size to Bozinovic and Srihari's.

They also state that the correct word occurs in the top two candidates in 81% of cases with the 710-word dictionary and 64% of cases with the 7800-word dictionary. (Elsewhere they state that these figures are for the correct word occurring in the top four candidates. It is not clear which number is correct.) Table 7.2 indicates the incidence of the correct word occurring in the top *n* word candidates, where *n* equals one, two, three or five, for the method presented here. These experiments use the 'root of product' basic score, the third scoring strategy and a character candidate certainty threshold of 0.075.

Dictionary Size	Top One Accuracy	Top Two Accuracy	Top Three Accuracy	Top Five Accuracy
112	95	97	97	97
710	85	89	94	94
7827	77	81	81	86

Table 7.2 "Top *n*" word recognition accuracy for *n* equal to 1, 2, 3 and 5. Accuracy is the number (or equivalently the percentage) of test words which are correctly placed in the top *n* word candidates by the matching method.

These experiments clearly indicate that the method presented in this chapter is significantly more accurate than Bozinovic and Srihari's technique. However, a precise comparison is difficult because the test data used is very different. Further testing of both methods on a larger, standard handwritten word database would be desirable. Figure 7.4 shows examples of words which are correctly recognized by the method and figure 7.5 gives examples of incorrectly recognized words.

The main causes of inaccuracy in these examples are degradation of the characters (e.g., in "BRUSSELS" and "ZURICH") and too much or too little over-segmentation (e.g., the 'W' in "GLASGOW" and the 'M' in "MALAGA" are spread across three segments, and the last 'D' in "MADRID" and the 'E' in "PRAGUE" are not separated from their preceding characters). Degradation causes greater ambiguity in the character classifications which causes further ambiguity in the word matching. Errors in over-segmentation mean that the desired character segmentation is not constructed, so it cannot be matched. Other problems are due to merged letters (e.g., the 'LA' of "MALAGA" and the 'RO' of "VERONA") which the over-segmentation method is not designed to separate. Some improvement could be attained by fine tuning the over-segmentation or extending it for the processing of merged letters. However, degradation of the images cannot be avoided. Some ambiguity of classification is

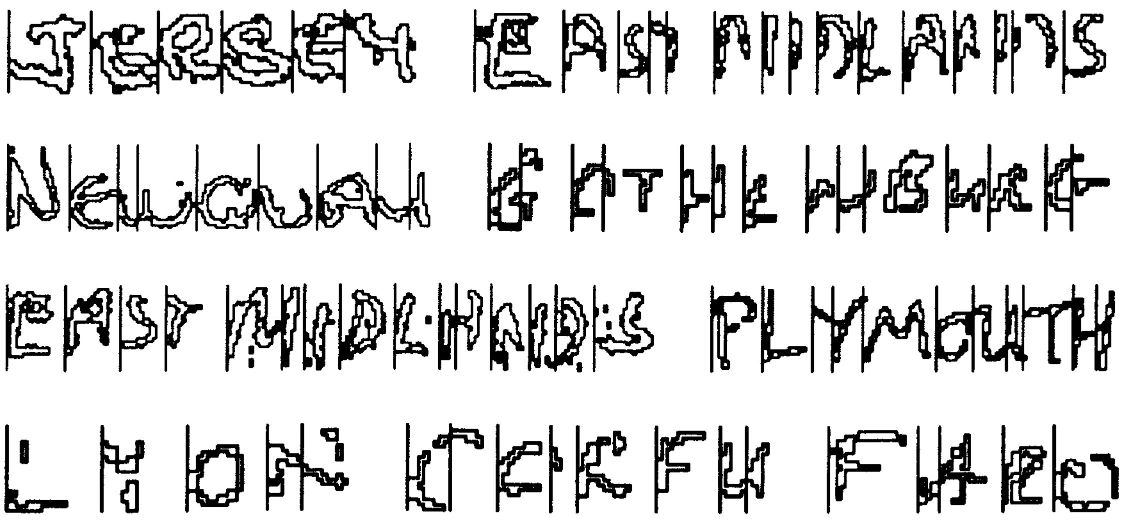


Figure 7.4 Examples of correctly recognized word outlines using the 'root of product' basic score, the third scoring strategy, character candidate certainty threshold 0.075 and the 7827-word dictionary. The vertical lines indicate the over-segmentation of the words.

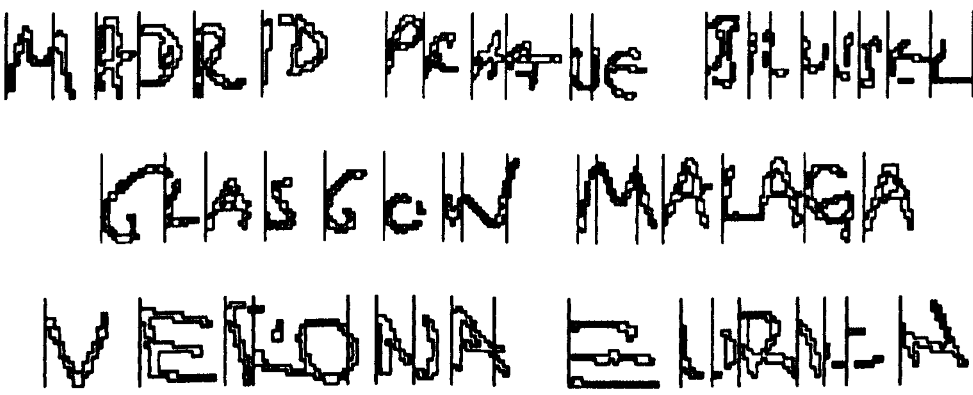


Figure 7.5 Examples of incorrectly recognized word outlines, using the same parameters as in figure 7.4.

therefore inevitable. Further contextual processing will be needed to select the correct word candidates.

The next set of tests investigates the number of word candidates produced by the method in relation to the character candidate certainty threshold. In the early experiments with agrep a low certainty threshold was used for the character candidate sets. Any character with a certainty (pseudo probability) greater than 0.05 was included in the candidate set for its segment group. In the later experiments this threshold was increased to 0.075 to reduce the number of character candidates and in turn reduce the size of the word candidate sets. The results in table 7.3 show the effect of varying this threshold on the average number of word candidates generated for each word and the recognition accuracy. All these tests use the 'root of product' basic score and the third scoring strategy.

Certainty Threshold	Average Number of Word Candidates for Dictionary Size:			Correct Words (%) for Dictionary Size:		
	112	710	7827	112	710	7827
0.050	28.81	166.19	1713.41	93	84	73
0.055	22.30	129.95	1331.53	93	84	73
0.060	16.76	99.21	1013.43	93	85	74
0.065	12.85	76.64	774.65	94	85	74
0.070	9.82	59.44	594.28	95	85	76
0.075	7.67	45.83	436.07	95	85	77
0.080	6.92	40.27	374.88	96	85	74
0.085	6.66	37.99	367.72	93	87	75
0.090	7.19	42.68	411.39	88	83	71
0.095	8.09	48.73	475.37	88	83	72
0.100	9.35	57.42	556.14	86	80	68

Table 7.3 Comparison of character certainty thresholds.

Note that as the threshold increases the accuracy remains roughly the same up to a certain point and then starts to decline. This is because the correct character classifications are being left out of the character candidate sets. Also note that increasing the threshold initially reduces the size of candidate sets but after a certain point the size begins to increase again. This is because more "wild cards" occur with larger thresholds. It is more likely that there will be no characters of sufficient certainty so all characters are allowed. "Wild cards" increase the chances of finding matches in the dictionary and hence increase the number of word candidates.

Values of around 0.080 offer the best results with the fewest candidates. The best accuracy on the 7827-word dictionary is achieved with a threshold of 0.075. However, the number of word candidates is still very high for large dictionaries.

Chapter 8 proposes a method for providing contextual feedback to an adaptive segmentation system (see section 5.7) which should be able to produce similar accuracy with far fewer candidates. The adaptive segmentation must initially be provided with a reasonably accurate "first guess" character segmentation. The next section suggests a method for producing that first attempt, based on the over-segmentation and character segmentation construction approaches of sections 7.3 and 7.4.

7.7. Choosing an Initial Character Segmentation

Section 7.4 described the construction of all possible legal character segmentations for the over-segmented words. An initial character segmentation, from which to start an adaptive segmentation process, can be selected from this set of legal possibilities by scoring each of them and choosing the one with the highest score. Section 7.5 suggested three strategies for scoring matched words. The best strategy involved taking the *n*th root of the product of the *n* character certainties and dividing by the average absolute difference between the segment group widths and the average segment group width. A similar strategy can be used to select a character segmentation, without actually matching the character sequences against a dictionary.

For each segment group, the certainty of the most certain character candidate is selected to represent the overall certainty of that segment group. Each legal character segmentation is then scored by taking the *n*th root of the *n* overall segment group certainties, and then dividing by the average absolute difference between the segment group widths and the average segment group width. Thus a score is determined in the same way as for matched words, but with the assumption that the highest certainty character candidate in each segment group is the correct one.

This strategy selects character segmentations which are close to the desired ones, but obviously cannot be certain of choosing the correct one in every case. Because the initial character segmentation may be inaccurate, and also because fewer character segmentations will be matched with this approach, it is likely that the dictionary matching will have to allow for the possibility of inserted and deleted characters. This will in turn require the word scoring method to be extended, to take insertions and deletions into account.

The effectiveness of the strategy at selecting initial character segmentations which are close to the desired ones has been tested using agrep (see section 7.5) to approximately match the initial character candidate sequences. This use of agrep allows insertions and deletions in the match, as well as substitutions. For each of the 100 test outlines, the number of substitution, insertion and deletion errors occurring in an approximate match between the initial character candidate sequence and the correct word was recorded.

In the word matching algorithm used to obtain the results in section 7.6, a word was accepted as a candidate if it was matched with a number of errors less than or equal to half the number of characters in the character segmentation. Note that only substitution errors were allowed in this algorithm. Using the same criterion for the approximately matched initial segmentation it was found that the correct word was accepted as a candidate in 66% of cases. By allowing one additional error (because we must expect more errors when insertions and deletions are allowed) this was increased to 82%. In other words, the initial segmentations were accurate enough for the correct word to be approximately matched in 82% of cases.

Figure 7.5 shows examples of some initial character segmentations obtained using this strategy. The original over-segmentation lines which are not included in the initial selection are indicated by dashed lines. It is clear from these examples, and from the above results, that the strategy is highly effective at choosing a good "first guess" segmentation in the cases which are suitably over-segmented. The cases where the correct words are not accepted as

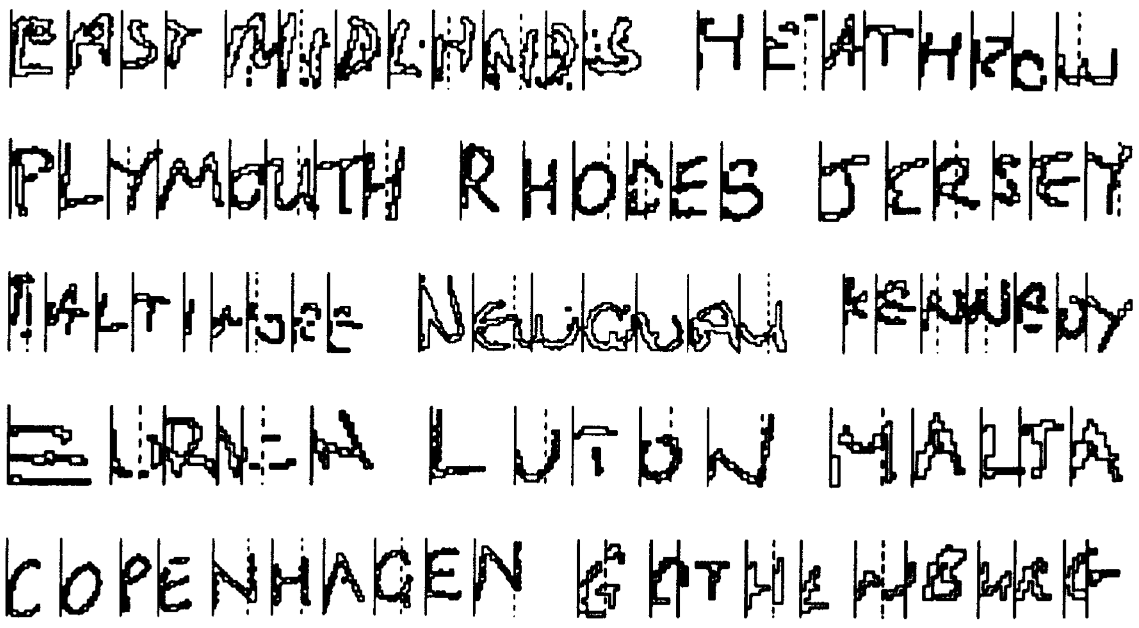


Figure 7.5 Examples of initial character segmentations. The solid vertical lines show the selected divisions. The dashed lines show the remaining over-segmentation lines which are not used in the initial character segmentation.

candidates are mainly due to inaccuracies in the over-segmentation, caused by degradation of the image or merged characters. Despite the effectiveness of the "first guess", the adaptive segmentation is still necessary because even with suitably over-segmented words, incorrect guesses are occasionally made.

7.8. Conclusions

A simple vertical segmentation system for non-merged handprinted characters has been presented which divides characters into either one or two segments (and very occasionally more than two). A method for grouping these segments to form characters and generating every possible sequence of these segment groups has been described. Each segment group is classified to a candidate set of characters using the new feature presented in chapter 5. Techniques for approximately matching pattern strings of these candidate sets against dictionaries have been tested. A simple but effective matching and scoring system has been developed which achieves good handprinted word recognition rates on a 100-word test set.

The main drawback of the method is that it is very slow. Although it could be made faster using a more complicated matching algorithm, it will still be slow because it searches every possible character segmentation. One way to reduce the search time is to reduce the number of legal character segmentations by improving the vertical segmentation system. Many of the divisions made by the system are incorrect (they occur in the middle of a character). If a way can be found to remove some of these incorrect divisions, without also removing the correct ones, then there will be fewer segments and hence fewer character segmentations. This is particularly important if the approach is extended to segment merged characters and allow them to fall across three segments, rather than the current two. Allowing for three-segment characters will cause an increase in the number of character segmentations which must be matched.

The number of legal character segmentations might also be reduced by using the character height and width information to eliminate patterns, rather than just to scale the scores of words matched by those patterns. It was noted in section 7.5 that, for handprinted words, the character widths in a character segmentation should be roughly equal. This led to the third scoring strategy which was very effective at finding the correct word. Instead of scaling down the scores of words with uneven character widths, a range of allowable deviation from the average width could be determined. Character segmentations with segment group widths outside this range could be removed from consideration. Height constraints might also be used, to prevent two characters being grouped as one.

Another way to improve the speed of the method is to implement the matching of each possible segmentation in parallel. A much faster serial approach would be the adaptive segmentation strategy (see section 5.7 and chapter 8) which starts with one legal character segmentation and is directed towards the correct segmentation by the higher level context. This will considerably reduce the number of patterns to be matched. Combined with a faster matching algorithm, this should give comparable results in a reasonable time. A method for selecting an initial character segmentation, from which to start an adaptive segmentation process, has been presented. This method is based on the scoring system for matched words but selects a good "first guess" character segmentation without performing any dictionary matching. Adaptive segmentation will probably require the word matching method to allow insertions and deletions of characters, as well as substitutions. It will then be necessary to extend the scoring system to cater for these insertions and deletions.

A further problem with the method is that it finds too many incorrect words and produces very large candidate word sets when a large dictionary is used. Again, this is because too many patterns are being generated. Improvements to the vertical segmentation or elimination of some character segmentations would reduce the number of patterns, as would the use of the adaptive segmentation strategy. It is probable, however, that several candidates will remain, and also that incorrect candidates will sometimes have higher certainties than the correct word. It must be concluded, therefore, that consideration of further contextual information is necessary to select the correct word from the candidate set.

In the airline ticket recognition problem, this further contextual information would come from other fields on the ticket, such as the airport identification code, the flight number and time, which could be individually recognized and matched against a database of flight information. Inconsistent candidates in each field would be eliminated until, hopefully, only one consistent match remains. Other applications may have similar fields which must all be consistent, e.g., in cheque recognition the amount in words must match the numerical amount. In general applications, additional context would come from a grammatical and semantic analysis of the surrounding text. Chapter 8 proposes a new classifier architecture for general contextual processing which can combine the adaptive segmentation, isolated character recognition and word recognition approaches along with higher level grammatical and semantic context.

7.9. Nomenclature

^	Start of line indicator.
$	End of line indicator.
DOT_SIZE	Outline size threshold. Placement of segmentation lines uses the thresholds LOOP_GAP1 and SEG_GAP1, or LOOP_GAP2 and SEG_GAP2, depending on whether the outline area is above or below this threshold.
fib(*n*)	*n*th element of the Fibonacci series.
LOOP_GAP1, LOOP_GAP2	Threshold distances between left-hand edge of an outline and the right-hand edge of the previous outline which determines whether or not a segmentation can be made.

n	Number of characters in a matched word.
p	Number of the current segment group structure.
s	Number of single segments in an over-segmented word.
SEG_GAP1, SEG_GAP2	Threshold distances between left-hand edge of an outline and the previous segmentation line which determines whether or not a segmentation can be made.
sequences	$fib(s) \times s$ array of pointers to segment group structures. The $fib(s)$ rows store the legal character segmentations.
total_groups	Total number of segment group structures for an over-segmented word.

Chapter 8

Future Work: Hierarchical Adaptive Contextual Classification

8.1. Summary

This chapter presents a proposal for a general contextual classification system, towards which this thesis has been working. Although the implementation is beyond the time constraints of this project, earlier chapters have presented character recognition solutions designed both for use in this system and for the testing of it.

The proposed system uses a hierarchical network of modular classifiers, each concerned with a different area of contextual knowledge. The hierarchical layering approach to contextual sensibility is described in sections 8.2 to 8.4 and guidelines for breaking a problem down into contextual layers are given in section 8.5.

Section 8.6 presents a new feedback strategy for inter-layer communication in the hierarchical network, based on "intelligent" suggestion of alternative inputs. Section 8.7 suggests strategies for using this feedback effectively in the hierarchical network.

The system makes greater requirements of classifiers than usual. Section 8.8 discusses these requirements and section 8.9 considers which specific classifiers are suitable. Section 8.10 presents a standard model of classifier modules which facilitates the use of different classification paradigms within the network.

Section 8.11 evaluates the proposed system in terms of the expected classification improvements and the difficulties of the approach and section 8.12 presents conclusions about the proposal.

8.2. The Contextual Layering Approach

Despite the obvious benefits to be gained from contextual information surprisingly little work has been published on a general theory of its use by classifiers. Most uses have been very application specific. Although the contextual information itself is application specific, the way it is used need not be. An approach to context-sensitive classification is proposed which aims to be general in nature, although certain restrictions are imposed on the classification paradigms which can be used.

In many classification problems, separate contextual levels can be easily identified. Each level becomes a classification problem on its own. At the lowest level is a noncontextual classification of each item of input data. Higher levels of context may be viewed as classifications of accumulated data from the lower levels. The proposed system is based on this notion of contextual levels. It provides a modular architecture which performs a complete contextual classification across the levels and allows each such problem to be tackled independently.

As an example of such a breakdown of context, let us return to the character recognition problem. The initial attempt might be to classify characters by their shape. The next level (the first real contextual level) might be a word recognizer which classifies ordered groups of characters as either valid or invalid words. Another layer might take ordered groups of words and determine whether they make syntactically valid sentences. Further layers might check for semantic context, possibly tailored for a specific domain (see section 6.6.4). (Another approach might start with subcharacter shapes which are initially classified into structural primitives; then groups of primitives are classified into characters. Other approaches might start with whole word shapes.) It would be beneficial if each of these levels could be dealt with separately, rather than trying to train a single classifier to learn them all.

Contextual sensitivity can be achieved through the use of a hierarchical network of classifiers. The problem is broken up into different context levels. Each level is handled by a suitable classifier. The original classification problem is then resolved by recombining the resulting classifiers in a hierarchical network. The different contextual layers of the problem feed into each other. Feedback from the higher-level classifiers is used to correct or adjust the lower levels to produce contextually sensible output.

Each level is modular in nature, can be trained separately and is independent of the rest of the hierarchy. Context modules can be added or removed at any time without the need to retrain the whole network.

Where the other input cases are independent of the target case, additional inputs must be obtained from another source. This chapter does not consider the problem of when and where to acquire this additional data but concentrates instead on problems where it is sufficient to base classifications on the context of the surrounding cases. For these problems the input may be treated as a stream and sampled through a sliding window. The current target case will usually be at the centre of the window. Classifications can now be based on the preceding and succeeding input cases as well as the target case. Note that this assumes a sequential ordering of the surrounding context. For non-sequentially ordered context, alternative windowing strategies can be used to group a target with its contextual neighbours.

Looking at several cases at once means using several instances of the lowest level classifier module, one for each input case in the window. There may also be multiple instances of the same classifier at the higher levels. However, each module performs most of its functions independently and so is highly suitable for a parallel implementation. This is fortunate as the more context we wish to consider the larger the window required at the input level. In the character recognition example, if we want to look at syntactic or semantic sensibility of sentences then we must have a window large enough to hold a whole sentence (made up of individual characters) at once.

The context to be considered may not fill the entire sliding window. In some applications that context may be delimited by *context markers.* Only the portion of the window between the context markers need be considered. An equivalent arrangement would be to use the context markers to resize the window of context which is considered. It is useful for the context markers to be treated as dummy input cases. This makes them simple to add, delete or move if their initial placement is incorrect. Also it is the simplest way to keep track of where they are: the markers get passed up and down the hierarchy along with the other input cases. This method fits logically into many applications; the breakup of context frequently corresponds to temporal or spatial gaps in the input stream.

In the character recognition example we may have a large window at the word recognition level, in order to handle long words. If the target input character is in a short word then the ends of that word must be indicated to the word level. Context markers are placed to represent the gaps between words. Note that this is merely an illustrative example; for character recognition it would actually be simpler to use the alternative method of resizing the window to exactly fit the context markers.

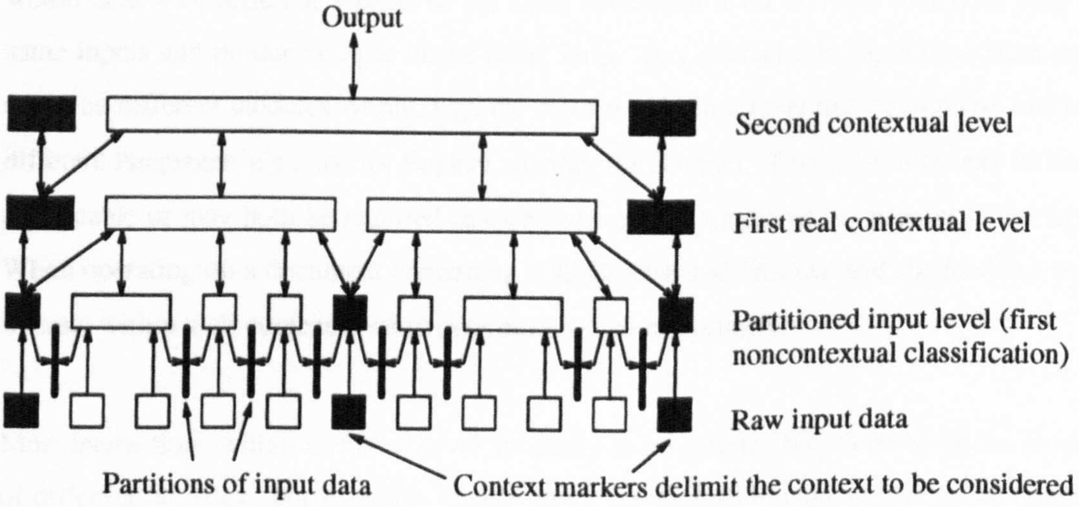

Output
Second contextual level
First real contextual level
Partitioned input level (first noncontextual classification)
Raw input data
Partitions of input data
Context markers delimit the context to be considered

Figure 8.1 Example arrangement of classifier modules and windows.

Figure 8.1 illustrates the arrangement for a very simple problem with two real contextual levels. A real system would probably have more levels and much larger context windows. In this case the context markers have been used to scale the windows to exactly fit the expected partitioning. The upwards arrows indicate the passing of classifications up the hierarchy. The downwards arrows indicate feedback which either alters the lower level classification or corrects the lowest level partitioning (see section 8.6).

To better understand the diagram we might consider it as if it were a real application, e.g., a character recognizer. Imagine that the lowest level elements are the over-segmented subcharacters (see section 7.3). The partitions are then the divisions of the legal character segmentation and the partitioned input elements are the isolated character classifications of segment groupings. The first real contextual level is a word recognizer module and the top level is a semantic sentence module.

The next two sections discuss the communication between classifier modules when they are in the same layer (intra-layer communication) and when they are in different layers (inter-layer communication).

8.3. Intra-Layer Module Communication

It may be logical and convenient to divide a level into yet more modules. Each module would deal with different aspects of the same contextual level but they would all take the same inputs and produce output of the same form. In a character recognition system there could be different modules within, say, the word recognition level that recognized words of different languages, e.g., one for English, another for German. These modules may be interchangeable or may both be required in cases where both languages are present in the input. When operating on a document containing both English and German text the modules must interact within their context level to determine which has priority.

Most interactions within a context level are likely to be simple choices between the outputs of different modules. For example, choose either the English module output or the German module output. Another strategy would be to form the inputs to the next level by concatenating the outputs of the multiple modules. More complicated combinations of classifiers may also be desired. Ho *et al.* [HHS90] take candidate output sets from multiple classifiers and form a subset of these which hopefully contains the correct class. The classes in the subset are then re-ranked according to a group consensus.

This chapter assumes intra-layer interactions can be reasonably easily resolved and for now only considers the interaction between levels. Each level is treated as containing only one type of module.

8.4. Inter-Layer Module Communication

It is theoretically possible for each classifier module to pass a probability for each possible output class to the next level. The higher level would then have all the information the lower level could provide about likely alternatives. There would be no need for feedback.

However, this is frequently impractical, except for very limited problems. In the character recognition example it would be unreasonable to output a probability for every possible word, or every possible sentence. A practical solution might be to pass only a small set of the most probable output classes, or to pass only one class at a time.

If a small set of outputs is passed to the next level then it becomes tempting to assume the correct output belongs to that set. The system could then ignore feedback. There will, however, be cases where the set does not contain the correct output and this approach will fail. A system where one class is passed at a time, and feedback is used to test alternatives, is not limited in the range of classes it can consider. A third possibility is a combination of the two methods: pass a small set of the most probable outputs and also use feedback to test alternatives that are not in the set. This is in essence the same as the one-at-a-time approach but aims to give an increase in speed by reducing the amount of feedback required.

A further consideration is the modularity of the context modules. It is not natural for a module to take a set of probabilities as its input. Rather it should take a representation of the input at the appropriate level of abstraction. For example, a word recognition module should ideally take a set of characters as its input, not a set of sets of probabilities of characters. The hierarchical classifier would therefore convert its input representation to a higher level of contextual abstraction at each stage, providing a more natural breakdown into modules. However, an associated confidence measure for each input is useful.

In the proposed system each classifier outputs one class at a time, along with a measure of its confidence. This measure is computed from the input values and other contextual knowledge, and may be based on probability theory, certainty theory, fuzzy logic, edit distance or whatever is logical for the particular type of classifier. Such a system must use a feedback mechanism to ensure contextual sensibility.

8.5. General Determination of Contextual Levels

The breakdown of contextual information into modules for the character recognition problem has been discussed above. The problem of contextual breakdown for general domains is largely intuitive, which is fortunate given the variety of forms which contextual information can take. The human brain processes contextual information from a vast number of its specialized areas and the many available sensory inputs. Abstracting these to a set of general rules is not yet possible; however, general guidelines can be given.

Wherever input to a classification problem comes from more than one separate source, one should consider passing each source's input to a separate context module before the information is combined at a higher context level. As an example, consider a robot guidance problem where information comes from both a visual source (camera focussed on the object) and an auditory source (microphones). It is logical for information from the two sources to pass through separate context modules, to obtain compatible representations and to identify classes strongly represented by either source in isolation, before attempting to combine the information. (In the human brain the sources would be processed initially by the visual and auditory cortices respectively before higher functions combine the input.) This is not always necessary and is sometimes more a modularization consideration than a contextual one.

When input cases are temporally or spatially ordered each case should be fed first to an instance of the bottom level context module. The outputs of these modules are grouped, and ordered context processing is performed by higher level context modules. For example, in speech recognition each lowest level primitive (usually a Fourier descriptor of the sampled sound segment) should be fed to a separate instance of the bottom level module (most probably a phoneme recognizer). The phonemes are then grouped together and the higher level context modules attempt to recognize higher forms, e.g., syllables, then words and then sentences (Bronkhurst *et al.* [BBS93]).

Once the first context level has been determined, further levels are best decided upon by identifying different levels of abstraction in the representation of the system output. Each such abstraction becomes the output from one contextual level, and the input to the next. The abstract representations should be ordered such that at each progressive level the amount of data required to store the representation decreases. Since the higher levels process a wider portion of the input, it is desirable to reduce the amount of data presented to these levels. Also it is almost always the logical order of abstraction.

There may be more than one contextual level between each adjacent pair of the abstract representations. For example, in character recognition there are at least two levels (syntactic context and semantic context) between the abstract representations, word and sentence. Determining the order in which to place these levels is problem specific.

These guidelines are not rigid and the breakdown is best determined by the needs of the problem. By keeping the required properties of the context module as simple as possible (see section 8.9), a large degree of flexibility is allowed in the determination and structuring of the contextual levels.

8.6. Intelligent Feedback

In cases where an initial classification attempt is contextually invalid it is necessary for higher level classifier modules to communicate the failure to the lower levels so that they might try again.

A brute force approach to this feedback would have each level take its set of inputs from the preceding level and compute its output and an associated confidence factor. If the confidence factor does not reach an *acceptance threshold* value then the next most probable set of output classes from the preceding level is selected instead, and the classification recalculated. The process would stop when the acceptance threshold was reached at each level, or when the initial confidence of a level's classification falls below a *rejection threshold* at which point the classifier concludes the input is unrecognizable and gives up. This approach would be very slow and a more intelligent feedback mechanism is required.

Rather than relying on the lower levels to suggest alternative classifications an intelligent feedback system would use the higher levels (which make use of wider context) to suggest contextually sensible alternatives. The higher levels direct the classification process towards a sensible result with much greater speed than an exhaustive search of the possible lower level classifications.

It is important, however, that the higher levels do not simply override the lower level results. Most approaches to contextual sensitivity so far have been a one-directional process, where each level passes on a contextually valid output to the next with no facility for verifying changes to the lower level classifications. These methods obtain a quick result but the final classification tends to be based on only a portion of the input data, the remaining portion having been rejected along the way. Just as the initial classification can give errors, so can the contextual checks. Mis-classifications inevitably occur and a system which can verify its contextual corrections, rather than just discarding contextually invalid input, would be more accurate.

Even when a higher level confirms the results from the lower levels it may not do so with a desirable level of confidence. A worthwhile aim of the hierarchical classifier is therefore not just to guess the correct output but to maximize the confidence of the classification. For some applications, the ones with which we are primarily concerned, the confidence could be improved by resampling some or all of the lowest level inputs. For example, in character recognition it might be improved by adjusting the segmentation of characters at the lowest level (a mechanism for such an adaptive segmentation process is described in section 5.7). Young and Matessa [YM92] used a similar approach for improving speech recognition.

In order for higher level hypotheses to be verified by the lower levels, and in order to propagate desired changes to the lowest level, it is necessary for each lower level's classification process to be reversed. Each level must be able to take a new output, fed back from the higher level, and work back to a representative input vector for that output. (The representative input should be such that when classified it will produce the new output with a high confidence.) This representative input vector can be compared to the actual input or can be fed back to the preceding level where the reversal process continues. Eventually this reversal process reaches the lowest level where the input data is resampled, this time with a better idea of what it is supposed to be. This resampling will usually involve a more accurate repartitioning of the raw input data which effects the neighbouring cases as well as the target case. The resulting improvement in lowest level accuracy propagates upwards to the uncertain classifications which can hopefully then be resolved.

In many cases there will be a conflict between what the lower level says the input is and what the higher levels say it ought to be. Most contextual approaches give priority to the higher level but it is desirable to verify such changes and in some cases keep the lower level classification.

In the hierarchical system neither level has priority unless the confidence of its output exceeds some threshold. If neither level is sufficiently confident then each level will maintain its classification, either until the surrounding contextual cases confirm one level as being correct (i.e., cause the other level's classification to change), or until all the contextual information has been considered and a decision must be made. If both levels produce an output confidence in excess of the acceptance threshold, but do not agree on the classification, or if neither level can satisfy the acceptance threshold and all the contextual information has been considered, then the classification with the highest confidence has precedence. In the event of a tie, priority is given to the higher level.

It is also desirable to use feedback to improve the confidence of a classification which is consistent with the next higher level's classification, but where neither level's output confidence exceeds the acceptance threshold. The mechanism for this is the same as for corrective feedback but rather than the higher level classifier suggesting an alternative input it suggests the same input. This is then passed back to the lower level as its output. The lower level converts it to a representative input vector (which would give a higher confidence classification at both levels). This target input propagates down to the bottom level which attempts to repartition, or otherwise adjust, the raw input data to better fit the target. The input is reclassified and the classification then propagates up the hierarchy again. If the feedback was successful, the confidence of the original result will improve.

8.7. Strategies for Fast, Accurate Classification

The individual classifiers are specified and then trained separately. The hierarchical network is formed by specifying connections between classifier modules. This involves choosing the sizes of sliding windows and ensuring that where two modules are connected the output of the lower module is of the same type as the input to the higher module (see section 8.10). This gives us a large network of classifiers with those dealing with the highest level of contextual abstraction at the top and those dealing with the raw data at the bottom.

Once the hierarchy has been defined, the problem is to find an order of classification, i.e., an order to use the modules and their functions, which is both fast and accurate. As ever, there is a trade off between speed and accuracy.

Since the top level provides the highest level of abstraction and has the widest view of the available context, the sooner that level is reached, the sooner the classification can be completed. Once the top level is satisfied, the lower levels will follow in due course. However, if the order of classification proceeds directly up the hierarchy, with no corrective feedback until it reaches the top, then the accumulated error will be greatest at that top level. The top level may then suggest a contextually valid solution based on erroneous classifications from the lower levels. The rest of the hierarchy will work towards the top level hypothesis which may lead them to enhance the previous errors. Classification will be fast but at the expense of accuracy.

The other extreme is to perform feedback at every opportunity to ensure that a high confidence, contextually valid classification is passed on at each stage. Even though these classifications are valid at one level, they may be found to be invalid at the next level of contextual abstraction, despite having a high confidence. The process must then be repeated between the lower levels. The relative lack of communication between the levels makes this method extremely slow, although it may eventually produce the most accurate classification.

A compromise is to proceed directly up the hierarchy to an intermediate level before introducing feedback. How far up this level should be depends on the expected error rate for each level. This error rate is problem specific so a general rule cannot be applied.

A fourth alternative is to introduce a *feedback threshold* for each level. After each attempted classification the feedback mechanism is used if the confidence measure is lower than the feedback threshold but greater than the rejection threshold. Thus if the initial confidence is high it is passed on to the next level without the system slowing down to check it, but if it is initially low (but not too low) the system tries to improve it before passing it on.

The feedback threshold would initially be somewhere between the acceptance and rejection thresholds but would increase with time and eventually equal the acceptance threshold. In the early stages the hierarchy quickly passes on high confidence classifications at each level and in the later stages it slows down (uses more feedback) to ensure the accuracy of its solution.

There is also the question of propagating high confidence results back to the lower levels (enforcing corrections or improving the lower level confidences if they agree with the higher level). This might be done the first time a classification confidence exceeds the classifier's acceptance threshold. Alternatively the system may wait until a conflict arises in the hierarchy and then back-propagate high confidence results in the vicinity of the conflict. Again there is a choice between accuracy and speed: the first method gives greater accuracy but is slow, and the second is fast but less accurate.

When considering speed limitations it should be borne in mind that all these strategies will benefit greatly from a parallel implementation.

8.8. Desired Properties of Classifier Modules

The hierarchical classification approach makes several demands of the classifier modules which make some classifiers unsuitable. Some others will require extensions to their abilities if they are to be used. This may seem to be a limitation of the hierarchical system but given the need for such a system it is really the classifiers that are limited, not the system. Classifier design has concentrated on the process of mapping the inputs to the output classes but not the feedback processes involved in context checking. An important area of research is the enhancement of classifiers for use in contextual environments.

The most significant criteria for determining whether or not a classifier is suitable for use within the hierarchical structure are speed, accuracy, generalization, ability to produce a confidence measure for the output and ability to produce a set of representative input vectors for each output class.

Speed of classification is essential because of the larger number of classifiers used, particularly at the lower levels, and the repetition of classification required by the feedback processes. A parallel implementation will make the system faster but each classifier will be used much more than in a one-directional system so fast classification and feedback are essential.

Accuracy and generalization are obvious criteria for any classifier but particularly in the contextual system. A high correct classification rate for the initial noncontextual classification will prevent the system from producing too many contextually valid but incorrect results. Contextual sensibility will be based on the initial classifications which have the highest confidence. Inaccurate results at this stage might have disastrous effects on correct but less confident results in the surrounding context.

The classifier must be able to produce some form of confidence measure (preferably a probability) of how well the output class represents the input in relation to other possibilities. Classifiers which can only produce a winner-takes-all result must be adapted, if possible, for this purpose, otherwise they are unsuitable. However, many such classifiers produce real-valued scores for each possible output class first before selecting the highest scoring possibility as the winner-takes-all output. For example, Bayesian statistical classifiers (see section 6.2) produce probabilities for each output class before choosing the most probable one. These values may easily be used as a confidence measure.

The most important property of suitable classifiers is their ability to produce intelligent feedback. When a higher level context module finds its input to be contextually invalid (when the output confidence does not exceed an acceptance threshold) the input is tentatively rejected by the classifier. Rather than simply informing the previous layer that it is incorrect, the classifier should suggest an alternative set of inputs which would produce a more confident (contextually valid) output. The preceding level receives this alternative *input* as a possibility for its *output*. In order to decide whether or not it is acceptable, this preceding level must be able to compare its real input vector with an input vector which is representative of the suggested output. This comparison will usually involve taking the Euclidean distance between the two vectors and comparing it to a threshold of acceptable similarity.

The key to the intelligent feedback mechanism is the conversion of a suggested output vector to a representative input vector. While other methods might be used to check suggested alternative outputs, the method of converting output to input is also required to propagate contextual corrections and adjustments down the hierarchy. It cannot therefore be avoided. Classifiers which are incapable of this conversion are unsuitable.

While many classifiers have the potential to operate in reverse, this ability is rarely ever used. Classification processes are almost always many-to-one functions from the inputs to the output. Mapping in the other direction would make a classifier much more powerful but a one-to-many mapping is not practical. It is required to find a one-to-one mapping from the set of output vectors to the set of input vectors such that the corresponding input vector is representative of the output class. For some problems this is extremely difficult to find but for most real world problems the input vectors form class-specific clusters in the input space and a vector from somewhere near the centre of a cluster might be an acceptable representative of the class. Where there are several clusters corresponding to the same class a central vector can be found for each and the one closest to the current input vector chosen.

Class representative vectors can be extracted from classifiers in two ways. Certain types of classifier (most statistical and some neural ones) explicitly represent the central vectors of clusters, e.g., as a mean vector for a class, or as a codebook vector in nearest-neighbour-based neural classifiers. Class representative vectors can be simply extracted from the parameters of these classifiers. The second, more complicated way can sometimes be used for classifiers which do not represent cluster centres explicitly but instead explicitly represent their decision boundaries. Since they (hopefully) form their decision boundaries around the clusters, the task of finding the centres of clusters becomes one of finding the centres of decision regions. However, finding the centre by examining the boundaries is extremely difficult, particularly for piecewise or multi-layered classifiers.

It may be desirable to find not just one vector in a decision region but a small set of vectors distributed across the region. A central vector gives no information about the decision boundaries which may be quite complex. While representing the boundaries exactly would require too many vectors a smaller number might be able to approximate them. A uniform distribution of the vectors across all classes would also give a better representation of the region.

In cases where it is not practical to extract vectors from the classifier itself there are two alternative approaches to determining class representative vectors. One is to pre-define a set of template vectors which are associated with the outputs. This requires the application to have a clearly defined set of "perfect" inputs. In practice this is rarely the case. The other approach is to determine the representative input vectors from the set of exemplar vectors used in training the classifier (but without referring to the classifier itself). Again, this involves finding class-specific clusters in the input space and selecting one or more vectors from each cluster. These vectors may simply be central vectors from the clusters or may be uniformly distributed across them.

8.9. Suitable Classifiers for Use as Classifier Modules

Many classifiers reviewed in chapter 6 have the qualities described above; however, some are unsuitable. This section discusses which specific classifiers may be used. The primary concern when selecting a suitable classifier is the ability to map outputs to representative input vectors.

Firstly, any classifier may be used if a set of "perfect" input templates can be assigned by hand or if class representative vectors are determined by clustering the training samples. Ideally though, we would like the classifier to determine the representative input vectors itself.

Parametric statistical classifiers usually include mean vectors for classes among their parameters. Nonparametric classifiers and statistical unsupervised learning methods which estimate the class conditional probability density functions will also usually estimate a mean vector for the class distributions. These mean vectors can be used as class representatives although we will normally require more than one representative per class. Piecewise classifiers are therefore most appropriate as means can be extracted for each cluster of a class represented by a discriminant function.

Clustering methods are generally very suitable as they encode multiple clusters of each class and (for well clustered data) should be accurate and have good generalization. It is assumed that we are dealing with data which naturally falls into clusters (otherwise it is unlikely that the feedback system will be effective). Cluster-based classifiers generally represent the cluster centres explicitly in their parameters so these can be easily extracted as class representative vectors. These types of classifier include the unsupervised statistical clustering classifiers and the competitive-cooperative neural learning models (additive and shunting networks, ART networks, learning vector quantization networks, radial basis function networks, self-organizing maps, counterpropagation networks). Although learning can be slow, these classifiers are usually fast in operation. Some of these methods do not produce confidence measures in their normal operation. However, they can usually be modified to produce measures based on the distances from the input vector to the cluster centres for each class.

Nonparametric statistical methods based on estimating discriminant functions are similar to the perceptron family of neural networks which learn by error-correction learning. This group of neural networks includes perceptrons, Adalines, Madalines, multilayer perceptrons, cascade correlation networks and Boltzmann perceptron networks. All these classifiers have similar problems in producing class representatives. The most powerful of these, the multilayer perceptron, is accurate and fast but provides no simple way to determine class representatives. Each neuron divides the input space with a simple hyperplane but decision regions are formed by combining the output of several neurons (using several hyperplanes to fence off the region). Finding the centre of such a region would be possible but the problem is further complicated by the use of hidden layers which allow for the creation of arbitrarily complex decision boundaries. Further investigation is needed to overcome these difficulties but for the moment it must be concluded that classifiers which encode decision boundaries are considerably less suitable for use in the hierarchical system than those which encode cluster centres.

Hebbian learning models (Hopfield nets, linear associative memories, bidirectional associative memories, learning machines and neocognitrons), and also those which combine Hebbian learning with stochastic learning (Boltzmann and Cauchy machines), face the same difficulties as error-correction learning networks. The weights do not directly encode any value which could usefully be used as representative input vectors. Some of these models only give a winner-take-all output and there is not always a logical way to produce confidence measures for other possible outputs. Another problem is that they generally have limited storage capacity which makes them unsuitable for representing large contextual domains.

Of the other neural networks described in chapter 6, the CMAC is a potentially useful classifier module. It is relatively easy to estimate the mean values of the inputs from the outputs which should enable class representatives to be determined. The CMAC learns arbitrarily complex mappings very quickly which is a useful, though not essential, quality for the hierarchical system. Its main disadvantage is its large memory requirement.

Structural classifiers are generally less accurate and much slower than statistical and neural classifiers but it is usually possible to determine confidence measures for each possible output class. Most structural representations, particularly the attributed variety, have matching (classifying) methods based on edit distances between inputs and templates. These edit distances are ideally suited to producing confidence measures for the outputs.

The simpler structural methods are also reasonably well suited to producing class representative input templates. Template matching and the basic string, tree and graph matching approaches use template libraries to store exemplars of classes. These may be used directly as representative inputs or might be reduced using vector quantization if there are too many of them.

The more complicated structural methods which use grammars to represent classes are not always suited to use in the hierarchical system. Although the grammars may be used to generate classes as well as match them, it is not easy to determine an "average" example of a class for use as a representative input. Most grammars consider each word in their language to be equally valid. An additional, external weighting of examples would be necessary if such a classifier was to produce contextual hypotheses as feedback. Stochastic grammars offer the possibility of weighting words so that "average" words for a class grammar can be extracted. Despite the difficulty of determining class representatives, structural grammars are potentially very useful for representing contextual domains.

Clearly there are many classifiers which might be used in the hierarchical network. To facilitate the use of all these different types within the hierarchical structure the next section proposes a standard model of a classifier module. If implementations of classifiers are tailored to fit this standard model then testing and usage of the hierarchical network will be much easier.

8.10. A Standard Model of a Classifier

Previous networks of classifiers have used the same classification paradigm for each classifier, e.g., Waibel *et al.* [WSS89], Green and Noakes [GN91] and Boyan [Boy92]. It is desirable to allow many different types of classifier to be used in the hierarchical framework; individual classifiers could then be chosen for their suitability to the particular task of their module.

In order for different classification paradigms to be used within the envisaged hierarchical network there must be a standard model which covers all classifiers (all those that can be used within this system). Each module should appear to the rest of the network as a "black box" where only the types of the inputs and outputs differ. All modules should operate in the same manner; a standard set of functions perform mappings between the inputs and outputs. The underlying paradigm and implementation should not affect the rest of the hierarchy.

Modules of the hierarchical classifier are defined to have an input vector and an output class with an associated confidence measure. (This definition can be stretched to allow the "output class" to be a set of classes, a vector or some other representation of the input data, appropriate to that module's level of contextual abstraction.) Usually a group of classifier outputs from a lower level are linked to the input vector of a single classifier at the next level. Each lower level output becomes an element of the higher level's input vector. Any classifier modules can be connected in this way provided the number of outputs matches the size of the input vector and the type of the output class is the same as the type of the input elements. This may require the addition of a simple non-classifying module to convert the output of one module into a suitable form for input to the next. Such non-classifying modules must be able to perform the same functions as other modules.

Each module has a function, *feedforward*, which maps its input vector to its output class, and another function, *feedback*, which maps from the output to the input. The *feedforward* function is the usual process of classifying the input and the *feedback* function is the internal feedback operation which maps output classes to representative input vectors.

Each module also has three thresholds specified. These thresholds are for comparison with the output confidence of the module's classification. A module must have an *acceptance threshold* above which a classification is accepted as correct at that level, a *rejection threshold* below which the classification attempt is abandoned until higher level contextual feedback can provide a sensible possibility (in other words the initial confidence is too low to be worth trying to improve), and a *feedback threshold* below which feedback is used to try to correct or improve the confidence of the result.

8.11. Evaluation

Future work on the hierarchical adaptive contextual classifier should be able to produce a system that will give greater overall accuracy than existing contextual processing techniques. The existing approaches have no equivalent to the inter-layer feedback of the hierarchical classifier. Since their methods can be formulated to fit the standard classifier model with little difficulty, the feedforward stage of the hierarchical classifier will be able to achieve exactly the same results as any other contextual system. It is expected that the addition of inter-layer contextual feedback will substantially improve on the feedforward results.

The intelligent feedback should allow more powerful searches to be conducted and should significantly increase classification accuracy above the levels attainable with feedforward-only systems. A flexible approach to the order of classification and feedback will allow the system to quickly find contextually valid classifications, and to verify them by re-evaluating their confidences. By directing the search for contextually valid solutions it will give greater speed than existing exhaustive search methods. The feedback will however increase the order of time complexity for the classification process over that of feedforward-only systems. The degree of increase will depend on the order of classification chosen.

Intelligent feedback should also allow incorrect but contextually valid solutions to be rejected if later classifications cause higher context levels to find the initial classification unacceptable. In feedforward-only systems this could only be done by overwriting the incorrect solution, with no facility for verifying the new solution, and with no information from the new solution contributing to the final top level result. The intelligent feedback system can both verify the correction and use it in the remainder of the classification process. It should therefore give much greater accuracy and confidence of results in cases where the early contextual stages are initially incorrect.

The verification of conclusions and the bottom level resampling of input data is expected to allow the system to produce results with greater confidence than existing context checking systems. Greater confidence at the lower levels also allows more sensible contextual choices to be made at the higher levels, giving a general improvement in system accuracy.

The general applicability and the modular nature of the hierarchical classifier should provide greater flexibility than existing systems, which are almost exclusively application specific. It should make alterations and extensions to the contextual information relatively simple to incorporate into an application; non-modular systems would be likely to require complete retraining.

In addition to the improved flexibility resulting from modularization, the standardization of classifier input/output and functionality is an important consideration for future designs. Classifier research has reached the stage where accuracy is comparable to human performance where no contextual information is available. Progress now requires advances in general contextual processing. Such processing need not be specific to either the application or the classifiers involved, but cannot be easily developed or tested without examples of them. This, at the moment, is the main handicap to the modular system: many existing classifiers need alterations before they fit the proposed standard model. Standardization of classifier design would therefore greatly facilitate research in this field.

The main cause of inaccuracy in the hierarchical network is expected to be false corrections to the early context levels. The introduction of some new errors is inevitable but it is believed that the number of accurate corrections will outweigh the false ones and so an overall increase in accuracy will be achieved. The incidence of new errors will depend on the emphasis placed on the contextual information in relation to the first non-contextual classification. This emphasis is determined by the acceptance and rejection thresholds for each context module. Errors are most likely to occur when there is little available context, e.g., in character recognition, when words (or sentences) are short. Such errors might be reduced by scaling the thresholds at each level according to the amount of context available (i.e., the number of elements in the input vector between the context markers).

The main weaknesses of the system are that it requires classifier modules to be capable of an approximate reverse classification. The proposed methods for this rely on classes forming a relatively small number of clusters in the input space. This is generally the case but there are some problems where this is not so and the intelligent feedback mechanism would not be suitable. The basic modularity and feedforward stages of the system would still be appropriate.

The hierarchical classifier is generally suitable for problems where contextual information is available and can be easily broken down into contextual levels. It is clearly unsuitable for domains where contextual information is limited or not available, e.g., fingerprint identification. The current model deals only with temporally and spatially ordered context in the initial input set and has not yet been extended for situations where contextual information, specific to the input case, must be gathered from additional sources. However, it is suitable for problems where the additional context can be learned independently in advance, i.e., when it applies to general cases rather than specific ones. The context of language is the most obvious example.

8.12. Conclusions

This chapter has proposed a hierarchical adaptive contextual classifier network for general contextual sensibility in classification. Contextual sensibility is essential if improvements are to be made in classifier accuracy; however, little work has been published on the general use of context in classifiers, beyond the inclusion of *a priori* probabilities. The ideas in this chapter therefore provide a very important direction for future research.

There is frequently a logical conceptual breakdown of context into different levels. A contextual layering approach is a valuable strategy for including a range of contextual information in a convenient and simple fashion. Windowing the layers' inputs allows sequentially ordered spatial and temporal context to be considered.

The proposed hierarchical classifier network is based on the contextual layering approach and can be used for a number of real world classification tasks such as character and speech recognition.

The hierarchical network's intelligent feedback mechanism requires its classifier modules to be able to reverse their classification process, i.e., map from an output class to a representative input. This method relies on the classes forming clusters in the input space but it is believed that this is true for most real problems. It has been shown that there are many classifiers which can do this easily. A future goal is to find reverse classification methods for other classifiers.

Strategies for ordering the system's consideration of the context have been devised. Further studies should evaluate the comparative effectiveness of these strategies and compare the generalized system to existing context-sensitive classification systems.

There is clearly much more work waiting to be done in this area. In particular, extensions to the feedback mechanisms should be explored. Currently when contextual corrections are verified they are enforced either by overwriting the results or by adjusting the bottom level segmentation. The overwriting enforcement technique is dangerous. More research is required to determine how confidence measures should best be increased to avoid creating over-confident classifications which may later be resistant to further correction from higher levels.

Chapter 9

Conclusions and Recommendations

9.1. Summary

This chapter reviews the original work presented in this thesis. Final concluding remarks are made about what what has been achieved and suggestions are made for future development of this work.

9.2. Conclusions

This thesis has presented several new developments in different areas of character recognition and proposed more efficient and accurate strategies for building full recognition systems. The most significant developments are a much improved vectorization algorithm, a highly robust feature for outline representation and an approximate dictionary matching method for recognizing over-segmented words. The strategies of minimal preprocessing and hierarchical adaptive contextual classification are also very significant new approaches for future character recognition research.

9.2.1. Outline-Based Vectorization

An outline-based vectorization method has been developed from Pavlidis's popular run-length-based algorithm, specifically for the vectorization of handwritten characters. Although slower than Pavlidis's method, it gives improved accuracy by considering lines made up of adjacent, vertical runs of black pixels as well as horizontal runs. Also several algorithms have been created to greatly improve the vectorization of junction areas. These include methods for identifying junction borders, removing "bow-tie" effects, estimating the direction of vector chains which lead up to the junctions, and determining the best choice of lines to join up the incoming vector chains.

The complete method has proved effective at producing accurate representations of the pen strokes used to draw characters, except where strokes overlap for long distances. Complete chains are used to represent strokes rather than the multiple, stick-like sections produced by

Pavlidis's method. Complete chains are much more useful for structural primitive extraction and are also closer to the kind of character representation used by on-line character recognizers. The possibility of using on-line recognition techniques on off-line images is very appealing because they are a great deal more accurate. However, the difficulty of determining the correct path of pen strokes through areas of overlap will probably prohibit a complete reconstruction and therefore keep the fields of on-line and off-line recognition distinct.

The weakness of vectorization, in general, is that it loses the geometric detail of character shape. This detail is essential for accurate recognition by statistical or neural classifiers. The results of comparing statistical classification of vectorized characters with their unvectorized outlines have shown that vectorization loses distinguishing detail or introduces errors, or both. A drop in generalization accuracy of between 7 and 16% occurred as a result of vectorization. The future usefulness of vectorization depends on accurate structural classification methods being developed, or on the successful application of on-line techniques to the vector chains.

9.2.2. Preprocessing

The difficulty in structural classification is in accurately extracting primitives on which to base the recognition. This was discussed, along with other preprocessing techniques, in chapter 4 and it was concluded that errors in extraction were inevitable. Correct classification therefore requires a substantial amount of error tolerance which causes too much class overlap and ambiguity in structural representations of characters. Similar problems occur with the extraction of polygonal approximations as primitives and it was concluded that a statistical or neural, feature-based classification is more appropriate for character recognition.

Of the other preprocessing techniques, it was found that smoothing, filling and joining lose important shape detail in many characters because they cannot distinguish between detail and the noise and distortion which they are intended to remove. These operations are detrimental to recognition in many cases and a better approach is to incorporate tolerance to noise and distortion in the later classification stages.

Line-width normalization, and slant and slope correction by shearing can deform the original characters and adversely effect recognition. Slant correction is still useful as an aid to vertical line segmentation but it can over-correct characters which are naturally slanted, e.g., '/' or '7'. It is most effective when applied to whole words rather than individual characters as this usually preserves natural slant. Slant and slope correction by rotation, and size and position normalization are non-deforming provided they do not transform to discrete coordinates. These techniques are beneficial to recognition but their effects can often be achieved more efficiently by later processing stages such as feature extraction. For example, it is possible to extract features which are invariant to size and position; separate stages for size and position normalization are therefore unnecessary.

A minimal preprocessing approach has been proposed which uses statistical or neural, feature-based classification in preference to structural classifiers and intends to find features which achieve the effects of the useful normalization operations. All stages should be tolerant to noise and distortion, and large training sets should be used to represent the variety of character styles.

9.2.3. The Radial Distance/Sector Area Feature

In the area of feature-based classification, the new Radial Distance/Sector Area feature has been presented for the representation of outlines to the classifier. This is invariant to size and position, approximately invariant to rotation, is fast to extract and is significantly more tolerant to noise than other outline-based features. Its most important quality is its robustness. It can operate on multi-part and broken character images which other outline-based features cannot. Excellent recognition rates have been achieved on isolated characters of the NIST and CEDAR databases.

The nature of the feature makes it possible to partially reconstruct the original shape from the feature values. This property can be used to correct the initial segmentation based on expected values of the features. The difference in values between the actual and expected features of a character can be related to differences in the actual and expected shape of the image representation. The segmentation of a character can therefore be adjusted to fit the expected shape. This adaptive segmentation has great potential for use in future development of the proposed hierarchical classification network, where expected feature values are determined according to contextual sensibility.

9.2.4. Integrated Contextual Segmentation

An approach to the integration of segmentation with contextual processing has been explored. A simple vertical over-segmentation system for non-merged handprinted characters has been developed and an algorithm for constructing legal sequences of possible character candidates has been presented. Techniques for approximately matching these sequences against dictionaries have been tested. A simple but effective matching and scoring system has been developed which achieves good handprinted word recognition rates on a 100-word test set.

The scoring system, based on character candidate certainties, is highly effective at selecting the correct word from large candidate sets. It is believed that by using an adaptive segmentation approach the search space can be considerably reduced without losing the correct word, thus accuracy can be preserved at greater speed. A method has been developed for finding a good initial segmentation from which to start an adaptive segmentation process.

Although the method achieves high accuracy results on small dictionaries, it is inevitable that as the size of dictionary increases, incorrect words will be matched which score higher than the correct word. The correct word will still rank in the top few word candidates but further contextual processing is required to select it from the candidate set. Generally, this additional context should come from consideration of the syntax and semantics of the surrounding text.

9.2.5. Hierarchical Adaptive Contextual Classification

A powerful new approach has been proposed for solving general contextual classification problems. Its implementation is beyond the time constraints of this research; however, the groundwork has been laid for building and developing the system and testing it on the character recognition problem. It is believed that hierarchical adaptive contextual classifiers will produce high accuracy classifications on a range of contextual problems, such as speech, map and engineering drawing recognition, and in particular off-line handwritten character recognition.

The most powerful innovation in the hierarchical network is the use of intelligent feedback to direct the search to the correct classification. A directed search is advantageous in practical applications where a blind, exhaustive search is usually too time consuming. The feedback also allows verification of contextual hypotheses and correction of the lower classification levels, including the lowest level sampling of the input (adaptive segmentation). This overcomes the limitation on almost all character recognition methods that they are restricted by the initial division of the input data. It also retains lower level information throughout the classification. This information is usually overwritten in other contextual approaches so that the final classification is based on only a portion of the data. Preservation of this information should allow greater accuracy and confidence of correctness in the hierarchical network's classifications.

The intelligent feedback requires each classifier to be able to reverse its classification and produce representative input vectors for hypothesized classes. It has been shown that many types of classifier can be relatively easily modified to extract these representatives from their internal parameters, provided the classes form clusters in the input space. It is believed that this is the case for most real world problems.

Further advantages of the proposed system are its modularity and flexibility. Classifiers are treated as black boxes and so a mixture of classifier types may be combined easily in the network. Classifier modules can be developed independently so alterations, extensions and retraining can be easily achieved without affecting the rest of the hierarchy. Non-modular systems would probably require complete retraining of the whole system.

Modularity requires standardization and there is currently very little standardization in the field. This means that existing classifiers require alterations to their input and output formats, and sometimes their operation, in order to fit the standard model. It is hoped that future research into general contextual processing systems will encourage the development of standards in character recognition and other domains.

9.3. Suggestions for Future Work

The handwritten character recognition problem is a long way from an effective solution and there is a great deal of scope for further developing the ideas presented here. This section discusses possible directions for improvement and extension of the methods and approaches presented in this thesis.

9.3.1. Outline-Based Vectorization

The outline-based vectorization method still suffers from the usual problem of thinning and vectorization algorithms. It is extremely difficult to accurately determine the paths of pen strokes where they overlap. The methods developed so far are effective for crossings of strokes (junctions) and short stubs where strokes double back on themselves. For long sections where strokes double back there is still much room for improvement. Currently the overlapped area is only likely to be crossed once by the vector representation. The width of the area may make it seemingly indistinguishable from that produced by a single, non-overlapping stroke. One approach to identifying and correcting these misrepresentations might arise through tackling another potential development.

Useful information might be obtained if the correct ordering and direction of the strokes can be determined. This is a particularly difficult task but if successful it could allow the more accurate on-line recognition techniques to be applied to off-line images. The aim is to recreate the sequence of pen movements and pen-up, pen-down actions used in drawing the characters, such as would be obtained by an on-line handwriting input device.

Few clues are available in an off-line image but certain heuristics can be applied which can help the determination of stroke direction. There is a general left-to-right trend in English cursive script. If only one stroke crosses a vertical pixel column it can be assumed that it crosses from left to right. Occasionally this assumption is false, for example, when the descender of a 'g' or 'y' extends out to the left. The method must be able to correct these choices later if they are wrong. Let us call these points on the vectorizations *direction markers.*

Since the outline-based vectorization method produces complete vector chains of strokes, it should now be possible to determine the direction of most strokes by following the chains and choosing the direction which is consistent with the direction markers. There will inevitably be some chains left over which are inconsistent. There are two possible reasons for this. One is that an incorrect assumption has been made about the direction of the stroke at the markers. This will usually occur when an ascender or descender is protruding out from the edge of an outline in a leftwards direction. Such cases should be relatively simple to identify and correct by reversing the direction of the marker. The other reasons is that the path of the pen in the original vectorization has been incorrectly determined. In most cases, this will be a result of the misrepresentation of overlapping strokes. This could therefore provide a means of identifying these regions, which could then be re-vectorized with the assumption that a stroke overlap has occurred.

These developments will not be easy to accomplish. It seems likely that both problems must be tackled in order to solve either one. There are still many remaining difficulties in recovering the on-line information. The time ordering of strokes can only be guessed at and there will probably be strokes which do not join onto any markers and so their directions must be guessed too. However, if these problems could be solved with high accuracy it would be a very significant step forward for character recognition.

9.3.2. Minimal Preprocessing

The minimal preprocessing approach requires little development. The main form of preprocessing which is beneficial and cannot be achieved in other ways is slant correction. It is advantageous to perform correction on whole words because it then aids their segmentation (assuming a vertical line segmentation). Correction based on the average slant of a word rather than the slant of a single character is also less prone to error. Development of a character-based feature which is invariant to slant is therefore not advisable. There is, however, room for improvement in word slant correction methods, particularly in accurately determining the angle of slant. Bozinovic and Srihari's method [BS89] looks to be the most promising approach (see section 2.3.5).

9.3.3. The Radial Distance/Sector Area Feature

The Radial Distance/Sector Area (RD/SA) feature loses information inside the outer boundary of characters. Often there are lines in the middle of characters which could provide useful discriminatory information which is currently lost by the Radial Distances. The Sector Area attempts to recover some of the information about the interior of the shape by looking at the distribution of its area, but if more detail is required then a second set of Radial Distance features could be taken.

The first possibility for these extra features is that they would measure the second furthest distance from the reference centre to a point on the outline, along each of the radials. Further feature sets could measure the third furthest distance and so on. The second furthest distances are probably very similar to the first furthest distances since they are usually measured from the near side of the outermost strokes. The third furthest distance would be the far side of the first interior stroke and so is probably more useful as a feature. The addition of these measurements should improve the recognition rate.

Another possible development of this feature approach is to make it faster to compute in an adaptive segmentation system. This would mean making it possible to rapidly compute the feature from its previous value, based on the change in segmentation. It is unlikely that this can be achieved for the complex segmentations described in section 5.7, but for simpler, vertical line segmentations it might be possible. The changes in segmentation would be small shifts in position of the vertical lines so changes in a segment's features might best be determined by looking at the pixels in the added and deleted vertical columns. This development would be limited to applications where vertical line segmentation is sufficient for correct segmentation of characters.

A variant of the feature which measures horizontally from a vertical reference line, rather than radially from a point, might allow fast recalculation based on the addition or removal of vertical pixel columns. The changes from the standard RD/SA feature would be that the reference centre is replaced by a vertical line through the centre of the character, the radial lines become equally spaced horizontal pixel rows, the radial distance becomes the distance to the furthest black pixel on the measurement row (two measurements: one to the left and one to the right) and the sector area becomes the number of black pixels between the measurement rows. It is likely that recognition accuracy using this variant would be close to the standard RD/SA feature. The difficulty in computing it from its previous value is in allowing for normalization of the values and movement of the central reference line when the segmentation changes.

9.3.4. Integrated Contextual Segmentation

The integrated contextual segmentation approach can be developed in two main ways. One way is to develop it for use with adaptive segmentation in the hierarchical network. The other is to develop it on its own as a feedforward-only system.

In either case, the next step in the development of the approach should be to devise an over-segmentation technique which handles both merged and non-merged characters. This technique should ideally divide characters into not more than three segments. The legal character segmentation construction method must then be extended to allow three-segment characters.

Developing the method for adaptive segmentation will probably require the approximate dictionary matching method to allow insertions and deletions of characters. The agrep method would be suitable for this purpose. The word scoring system would then need to be extended to take insertions and deletions into account.

If the method is to be used without adaptive segmentation then an important aim is to reduce the number of legal character segmentations which have to be matched against the dictionary. This reduction can be based on the widths, and possibly heights, of segment groups in relation to the average segment group width of the character segmentation. Character segmentations containing segment groups whose measurements deviate too greatly from the average should be eliminated. Hopefully the number of character segmentations can be considerably reduced without eliminating the correct one.

9.3.5. Hierarchical Adaptive Contextual Classification

The hierarchical adaptive contextual classification network is clearly a very important area for future research. Accurate contextual processing is essential if handwritten character recognition is to progress to practical levels of accuracy. The hierarchical network incorporates several sensible ideas for achieving this aim and has great potential as an approach to the problem. Suggestions for general development of the approach have already been discussed in chapter 8. This section discusses further the testing of the network and its development for a character recognition application.

Initial testing and development would benefit from some simplification of the approaches described in earlier chapters. The complex segmentation of cursive words, detailed in section 5.7, could be replaced with a vertical segmentation to simplify the adaptive segmentation layer. Vertical segmentation would benefit from slant correction, as mentioned in section 9.3.2. The Radial Distance/Sector Area feature should be modified as described in section 9.3.3 to facilitate the rapid recalculation of feature values during vertical adaptive segmentation. The isolated character classification module would be a radial basis function classifier using these features, as described in section 7.4. An initial segmentation of words can be determined from a set of over-segmented sections using the strategy presented in section 7.7. The word recognition module would use the approximate dictionary matching system of section 7.5. This arrangement would provide a good starting point for testing the feedforward and feedback mechanisms of the network.

There are initially three key areas in developing the hierarchical network. Firstly, the strategies for ordering the feedforward and feedback processes must be further investigated. Several possibilities have been suggested and one has been proposed as the most effective strategy, but alternatives should also be considered. Secondly, the assignment of acceptance, rejection and feedback thresholds for classifier modules must be explored. Tests must be conducted to find optimal values for each module. Thirdly, the alteration of modules' output confidences as a result of contextual verification or correction must be investigated. The application of probability theory, certainty theory, possibility theory or other techniques to this alteration should be explored.

A final consideration is that the modularity of the approach is an important aid to development. The contextual mechanisms need not be concerned with the specifics of the application and modularization provides a way of hiding these application specific details from the network. Future research in contextual processing can then concentrate on the arrangement of modules and the communication between them. Modularization also allows the continued development of individual classifiers for specific stages of recognition to be conducted independently. As noted in section 9.2, modularity requires much greater standardization than currently exists. The field of character recognition in general, and in particular the contextual processing area, would be greatly aided by a standardization of data formats for image representation, segmented image representation, features, classifier inputs and outputs, and further formats for higher levels of contextual abstraction (e.g., word output, sentence output). This should be given serious consideration in the near future.

References

[AAH91] Abe, K. , Arcelli, C. and Held, A. (1991), "Split and Merge for a Hierarchical Contour Sketch via Dominant Point Detection," *Proceedings of the 1st International Conference on Document Analysis and Recognition*, pp. 392-400, Saint-Malo, France.

[Abe68] Abend, K. (1968), "Compound Decision Procedures for Unknown Distributions and for Dependent States of Nature," in *Pattern Recognition*, ed. L. Kanal, pp. 207-249, Thompson, Washington, D.C., U.S.A.

[Agm54] Agmon, S. (1954), "The Relaxation Method for Linear Inequalities," *Canadian Journal of Mathematics*, vol. 6, pp. 382-392.

[AHF88] Assal, M.H.A., Horne, E. and Fairhurst, M.C. (1988), "An Enhanced Linear Model Edge Detector," in *Lecture Notes in Computer Science*, vol. 301, *Proceedings of the British Pattern Recognition Association 4th International Conference on Pattern Recognition*, ed. J. Kittler, pp. 68-79, Springer-Verlag.

[AHS85] Ackley, D.H., Hinton, G.E. and Sejnowski, T.J. (1985), "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol. 9, pp. 147-169.

[AKC90] Ahalt, S.C., Krishnamurthy, A.K., Chen, P. and Melton, D.E. (1990), "Competitive Learning Algorithms for Vector Quantization," *Neural Networks*, vol. 3, no. 3, pp. 27-290.

[Alb75a] Albus, J.S. (1975), "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Transactions of the ASME, Series G, Journal of Dynamic Systems, Measurement, and Control*, vol. 97, pp. 220-227.

[Alb75b] Albus, J.S. (1975), "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," *Transactions of the ASME, Series G, Journal of Dynamic Systems, Measurement, and Control*, vol. 97, pp. 228-233.

[Als88] Alstyne, M. v. (1988), "Remaking the Neural Net: A Perceptron Logic Unit," *Neural Networks Supplement: INNS Abstracts*, vol. 1, p. 143.

[AM88] Amari, S.-I. and Maginu, K. (1988), "Statistical Neurodynamics of Associative Memory," *Neural Networks*, vol. 1, no. 1, pp. 63-73.

[Ama67] Amari, S.-I. (1967), "A Theory of Adaptive Pattern Classifiers," *IRE Transactions on Electronic Computers*, vol. 16, pp. 299-307.

[Ama83] Amari, S.-I. (1983), "Field Theory of Self-Organizing Neural Nets," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, pp. 741-748.

[Ams82] Amsler, R.A. (1982), "Computational Lexicology: A Research Program," *National Computer Conference, AFIPS Conference Proceedings*, pp. 657-663.

[And68] Anderson, J. (1968), "A Memory Storage Model Utilizing Spatial Correlation Functions," *Kybernetik*, vol. 5, pp. 113-119.

[AP77] Ali, F. and Pavlidis, T. (1977), "Syntactic Recognition of Handwritten Numerals," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 7, pp. 537-541.

[Arc81] Arcelli, C. (1981), "Pattern Thinning by Contour Tracing," *Computer Graphics and Image Processing*, vol. 17, pp. 130-144.

[AS78] Arcelli, C. and Sanniti di Baja, G. (1978), "On the Sequential Approach to Medial Line Transformation," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 8, no. 2, pp. 139-144.

[AS80] Arcelli, C. and Sanniti di Baja, G. (1980), "Medial Lines and Figure Analysis," *Proceedings of the 5th International Conference on Pattern Recognition*, pp. 1016-1018, IEEE.

[AS81] Arcelli, C. and Sanniti di Baja, G. (1981), "A Thinning Algorithm Based on Prominence Detection," *Pattern Recognition*, vol. 13, no. 3, pp. 225-235.

[AS85] Arcelli, C. and Sanniti di Baja, G. (1985), "A Width-Independent Fast Thinning Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 463-474.

[AS89] Arcelli, C. and Sanniti di Baja, G. (1989), "A One-Pass Two-Operations Process to Detect the Skeletal Pixels on the 4-Distance Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 411-414.

[AU72] Aho, A.V. and Ullman, J.D. (1972), *The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing*, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.

[Aus88] Austin, J. (1988), "Grey Scale N Tuple Processing," in *Lecture Notes in Computer Science*, vol. 301, *Proceedings of the British Pattern Recognition Association 4th International Conference on Pattern Recognition*, ed. J. Kittler, pp. 110-119, Springer-Verlag.

[Bai88] Baird, H.S. (1988), "Feature Identification for Hybrid Structural/Statistical Pattern Classification," *Computer Vision, Graphics and Image Processing*, vol. 42, pp. 318-333.

[Bal81] Ballard, D.H. (1981), "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111-122.

[Ban91] Banks, R.N. (1991), *Neural Networks for Handprinted Character Recognition*, Ph.D. thesis, University of Nottingham, Nottingham, England.

[Bar88] Baruch, O. (1988), "Line Thinning by Line Following," *Pattern Recognition Letters*, vol. 8, no. 4, pp. 271-276.

[BB59] Bledsoe, W.W. and Browning, I. (1959), "Pattern Recognition and Reading by Machine," *Proceedings of the Eastern Joint Computer Conference*, pp. 225-232.

[BBM64] Bashkirov, O.A., Braverman, E.M. and Muchnik, I.B. (1964), "Potential Function Algorithms for Pattern Recognition Learning Machines," *Automation and Remote Control*, vol. 25, pp. 629-631.

[BBS93] Bronkhurst, A.W., Bosman, A.J. and Smoorenburg, G.F. (1993), "A Model for Context Effects in Speech Recognition," *Journal of the Acoustical Society of America*, vol. 93, no. 1, pp. 499-509.

[BC90] Barnard, E. and Casasent, D. (1990), "Shift Invariance and the Neocognitron," *Neural Networks*, vol. 3, no. 4, pp. 403-410.

[BCG81] Bezdek, J.C., Coray, C., Gunderson, R. and Watson, J. (1981), "Detection and Characterization of Cluster Substructure," *SIAM Journal of Applied Mathematics*, vol. 40, pp. 339-372.

[BD88] Broomhead, D.S. and Lowe, D. (1988), "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, vol. 2, pp. 321-355.

[Bea80] Beaumont, G.P. (1980), *Intermediate Mathematical Statistics*, Chapman and Hall.

[Beu73] Beun, M. (1973), "A Flexible Method for Automatic Reading of Handwritten Numerals," *Philips Technical Review*, vol. 33, no. 4, pp. 89-101, 130-137.

[Bez81] Bezdek, J.C. (1981), *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, New York, U.S.A.

[BF77] Brayer, J.M. and Fu, K.-S. (1977), "A Note on the *k*-tail Method of Tree Grammar Inference," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, pp. 293-300.

[BF79] Bentley, J.L. and Friedman, J.E. (1979), "Data Structures for Range Searching," *Computing Surveys*, vol. 11, no. 4.

[BG89] Baeza-Yates, R.A. and Gonnet, G.H. (1989), "A New Approach to Text Searching," *Proceedings of the 12th Annual ACM-SIGIR Conference on Information Retrieval*, pp. 168-175, Cambridge, Mass., U.S.A.

[BH67] Ball, G.H. and Hall, D.J. (1967), "A Clustering Technique for Summarizing Multivariate Data," *Behavioural Science*, vol. 12, pp. 153-155.

[Bha43] Bhattacharyya, A. (1943), "On a Measure of Divergence Between Two Statistical Populations Defined by Their Probability Distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99-109.

[BHN68] Bakis, R., Herbst, N.M. and Nagy, G. (1968), "An Experimental Study of Machine Recognition of Hand-Printed Numerals," *IEEE Transactions on Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 119-132.

[BK88] Baptista, G. and Kulkarni, K. (1988), "A High Accuracy Algorithm for Recognition of Handprinted Numerals," *Pattern Recognition*, vol. 21, pp. 287-291.

[BK94] Bose, C.B. and Kuo, S.-S. (1994), "Connected and Degraded Text Recognition Using Hidden Markov Model," *Pattern Recognition*, vol. 27, no. 10, pp. 1345-1363.

[BKP86] Baird, H.S., Kahan, S. and Pavlidis, T. (1986), "Components of an Omnifont Page reader," *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 344-348, IEEE, Paris, France.

[BL70] Block, H.D. and Levin, S.A. (1970), "On the Boundedness of an Iterative Procedure for Solving a System of Linear Inequalities," *Proceedings of the American Mathematical Society*, vol. 26, pp. 229-235.

[Blu67] Blum, H. (1967), "A Transformation for Extracting New Descriptors of Shape," in *Models for the Perception of Speech and Visual Form*, ed. W. Wathen-Dunn, pp. 362-380, MIT Press, Cambridge, Mass., U.S.A.

[Bor84] Borgefors, G. (1984), "Distance Transformations in Arbitrary Dimensions," *Computer Vision, Graphics and Image Processing*, vol. 27, pp. 321-345.

[Bow84] Bow, S.-T. (1984), *Pattern Recognition*, Marcel Dekker, New York, U.S.A.

[Boy92] Boyan, J.A. (1992), *Modular Neural Networks for Learning Context-Dependent Game Strategies*, M.Phil. thesis, University of Cambridge, Cambridge, England.

[Bre65] Bresenham, J.E. (1965), "Algorithm for Computer Control of Digital Plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25-30.

[BS82] Bozinovic, R.M. and Srihari, S.N. (1982), "A String Correction Algorithm for Cursive Script Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, no. 6, pp. 655-663.

[BS88] Borgefors, G. and Sanniti di Baja, G. (1988), "Skeletonizing the Distance Transform on the Hexagonal Grid," *Proceedings of the 9th International Conference on Pattern Recognition*, vol. 1, pp. 504-507, IEEE, Rome, Italy.

[BS89] Bozinovic, R.M. and Srihari, S.N. (1989), "Off-Line Cursive Script Word Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 68-83.

[BSA91] Belkasim, S.O., Shridhar, M. and Ahmadi, M. (1991), "Pattern Recognition with Moment Invariants: A Comparative Study and New Results," *Pattern Recognition*, vol. 24, no. 12, pp. 1117-1138.

[BT73] Booth, T.L. and Thompson, R.A. (1973), "Applying Probability Measures to Abstract Languages," *IEEE Transactions on Computers*, vol. 22, no. 5, pp. 442-450.

[Bur86] Burr, D. (1986), "A Neural Network Digit Recognizer," *Proceedings of the International Conference on Systems, Man and Cybernetics*, pp. 1621-1625, IEEE, Atlanta, U.S.A.

[Bur87] Burr, D.J. (1987), "Experiments with a Connectionist Text Reader," *Proceedings of the 1st IEEE International Conference on Neural Networks*, vol. IV, pp. 717-724, IEEE, San Diego, U.S.A.

[But93] Buturovic, L.J. (1993), "Improving *k*-Nearest-Neighbour Density and Error Estimates," *Pattern Recognition*, vol. 26, no. 4, pp. 611-616.

[BV75] Breuer, P. and Vajta, Jr, M. (1975), "Structural Character Recognition by Forming Projections," *Prob. Control Information Theory*, vol. 4, pp. 339-352.

[BW88] Baum, E. and Wilczek, F. (1988), "Supervised Learning of Probability Distributions by Neural Networks," in *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, pp. 53-58, American Institute of Physics, New York, U.S.A.

[Cal70] Calvert, T.W. (1970), "Nonorthogonal Projections for Feature Extraction in Pattern Recognition," *IEEE Transactions on Computers*, vol. 19, pp. 447-452.

[Car66] Carlson, G. (1966), "Techniques for Replacing Characters that are Garbled on Input," *1966 Spring Joint Computer Conference, AFIPS Conference Proceedings*, vol. 28, pp. 189-192, Spartan, Washington, D.C., U.S.A.

[Car89] Carpenter, G.A. (1989), "Neural Network Models for Pattern Recognition and Associative Memory," *Neural Networks*, vol. 2, no. 4, pp. 243-257.

[Cas70] Casey, R.G. (1970), "Moment Normalization of Handprinted Characters," *IBM Journal of Research Developments*, vol. 14, pp. 548-557.

[CC73] Caskey, D.L. and Coates, Jr, C.L. (1973), "Machine Recognition of Handprinted Characters," *Proceedings of the 1st International Joint Conference on Pattern Recognition*, pp. 41-49.

[CCI84] CCITT (1984), "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus," Fascicle VII.3 — Rec. T.6.

[Ced79] Cederberg, R.L.T. (1979), "Chain-Link Coding and Segmentation for Raster Scan Devices," *Computer Graphics and Image Processing*, vol. 10, pp. 224-234.

[CG87a] Carpenter, G.A. and Grossberg, S. (1987), "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54-115.

[CG87b] Carpenter, G.A. and Grossberg, S. (1987), "ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, vol. 26, pp. 4919-4930.

[CG88] Chieuh, T.-D. and Goodman, R. (1988), "A Neural Network Classifier Based on Coding Theory," *Proceedings of the 1987 IEEE Conference on Neural Information Processing Systems — Natural and Synthetic*, pp. 174-183, American Institute of Physics, New York, U.S.A.

[CH67] Cover, T.M. and Hart, P.E. (1967), "Nearest Neighbour Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21-27.

[CH89] Chen, Y.-S. and Hsu, W.-H. (1989), "A Systematic Approach for Designing 2-Subcycle and Pseudo 1-Subcycle Parallel Thinning Algorithms," *Pattern Recognition*, vol. 22, no. 3, pp. 267-282.

[Che52] Chernoff, H. (1952), "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on a Sum of Observations," *Annals of Mathematical Statistics*, vol. 23, pp. 493-507.

[Che65] Chen, C.-H. (1965), "A Computer Searching Criterion for Best Feature Set in Character Recognition," *Proceedings of the IEEE*, vol. 53, pp. 2128-2129.

[Cho57] Chomsky, N. (1957), *Syntactic Structures*, Mouton, The Hague, Netherlands.

[CKZ94] Chen, M.-Y., Kundu, A. and Zhou, J. (1994), "Off-Line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 481-496.

[CL85] Cheng, Y.S. and Leung, C.H. (1985), "Chain-Code Transform for Chinese Character Recognition," *Proceedings of the International Conference on Cybernetics and Society*, pp. 42-45, IEEE, Tucson, Arizona, U.S.A.

[CL88] Caelli, T.M. and Liu, Z.Q. (1988), "On the Minimum Number of Templates Required for Shift, Rotation and Size Invariant Pattern Recognition," *Pattern Recognition*, vol. 21, no. 3, pp. 205-216.

[CN82] Casey, R.G. and Nagy, G. (1982), "Recursive Segmentation and Classification of Composite Character Patterns," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 1023-1026, IEEE, Munich, Germany.

[Con89] Connor, P.J. (1989), "Outlines of Binary Images," *Proceedings of the 5th International Conference on Computer Aided Production Engineering*, Edinburgh.

[Cor93] Cornforth, D.J. (1993), *Classifiers for Machine Intelligence*, Ph.D. thesis, University of Nottingham, Nottingham, England.

[Cos60] Cosgriff, R.L. (1960), "Identification of Shape," Report 820-11, ASTIA AS 254 792, Ohio State University Research Foundation, Columbus, U.S.A.

[CP76] Casasent, D. and Psaltis, D. (1976), "Position, Rotation and Scale Invariant Optical Correlation," *Appl. Optics*, vol. 15, pp. 1795-1799.

[CP79] Chen, P. and Pavlidis, T. (1979), "Segmentation by Texture Using a Co-Occurrence Matrix and a Split-and-Merge Algorithm," *Computer Graphics and Image Processing*, vol. 10, pp. 172-182.

[Cri94] Cripps, M. (1994), "Prototyping Report for Match Quality," PAFEC Ltd Internal Report DTHC20_prot.

[CS83] Cottrell, G. and Small, S. (1983), "A Connectionist Scheme for Modeling Word Sense Disambiguation," *Cognition and Brain Theory*, vol. 1, pp. 89-120.

[CS84] Cottrell, G. and Small, S. (1984), "Viewing Parsing as Word Sense Discrimination: A Connectionist Approach," in *Computational Models of Natural Language Processing*, ed. B. Bara and G. Guida, pp. 91-119, Elsevier Science Publishers, New York, U.S.A.

[CS86] Chu, Y.K. and Suen. C.Y. (1986), "An Alternative Smoothing and Stripping Algorithm for Thinning Digital Binary Patterns," *Signal Processing*, vol. 11, no. 3, pp. 207-222.

[CT65] Cooley, J.W. and Tukey, J.W. (1965), "An Algorithm for Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, pp. 297-301.

[CT94] Cheng, B. and Titterington, D.M. (1984), "Neural Networks: A Review from a Statistical Perspective," *Statistical Science*, vol. 9, pp. 2-54.

[CWS87] Chin, R.T., Wan, H.-K., Stover, D.L. and Iverson, R.D. (1987), "A One-Pass Thinning Algorithm and Its Parallel Implementation," *Computer Vision, Graphics and Image Processing*, vol. 40, pp. 30-40.

[Dan80] Danielsson, P.E. (1980), "Euclidean Distance Mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227-248.

[Dan82] Danielsson, P.E. (1982), "Encoding of Binary Images by Raster-ChainCoding of Cracks," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 335-338, IEEE, Munich, Germany.

[Dav75] Davis, L.S. (1975), "A Survey of Edge Detection Techniques," *Computer Graphics and Image Processing*, vol. 4, pp. 248-270.

[Dav82] Davis, L.S. (1982), "Hierarchical Generalized Hough Transforms and Line-Segment Based Generalized Hough Transforms," *Pattern Recognition*, vol. 15, no. 4, pp. 277-285.

[Dav90] Davies, E.R. (1990), "Chapter 15 — Abstract Pattern Matching Techniques," in *Machine Vision: Theory, Algorithms, Practicalities*, pp. 345-368, Academic Press.

[Dea81] Deans, S.R. (1981), "Hough Transform from the Radon Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, no. 2, pp.

185-188.

[Dea83] Deans, S.R. (1983), *Applications of the Radon Transform*, John Wiley and Sons, New York, U.S.A.

[Den86] Denker, J.S. (1986), "Neural Network Models of Learning and Adaptation," *Physica*, vol. 22D, pp. 216-232.

[Des80] Dessimoz, J.-D. (1980), "Specialized Edge-Trackers for Contour Extraction and Line-Thinning," *Signal Processing*, vol. 2, no. 1, pp. 71-73.

[Deu69] Deutsch, E.S. (1969), "Comments on a Line Thinning Scheme," *Computer Journal*, vol. 12, p. 412.

[Deu72] Deutsch, E.S. (1972), "Thinning Algorithms on Rectangular, Hexagonal and Triangular Arrays," *Communications of the ACM*, vol. 15, no. 9, pp. 827-837.

[DH68] Duda, R.O. and Hart, P.E. (1968), "Experiments in the Recognition of Hand-Printed Text: Part II — Context Analysis," *1968 Fall Joint Computer Conference, AFIPS Conference Proceedings*, vol. 33, pp. 1139-1149, Thompson, Washington, D.C., U.S.A.

[DH72] Duda, R.O. and Hart, P.E. (1972), "Use of the Hough Transform to Detect Lines and Curves in Pictures," *Communications of the ACM*, vol. 15, pp. 11-15.

[DH73] Duda, R.O. and Hart, P.E. (1973), *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, U.S.A.

[Din55] Dinneen, G.P. (1955), "Programming Pattern Recognition," *Proceedings of the Western Joint Computer Conference*, pp. 94-100, New York, U.S.A.

[DK82] Devijver, P.R. and Kittler, J. (1982), *Pattern Recognition: A Statistical Approach*, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.

[DL90a] Dinstein, I. and Landau, G.M. (1990), "Parallel Computable Contour Feature Strings for 2D Shape Recognition," *SPIE Technical Symposium Opt. Engineering Photonics Aerospace Sensing (Real-Time Image Processing II)*, Orlando, Florida.

[DL90b] Downton, A.C. and Leedham, C.G. (1990), "Preprocessing and Presorting of Envelope Images for Automatic Sorting Using OCR," *Pattern Recognition*, vol. 23, no. 3/4, pp. 347-362.

[DLG91] Dinstein, I., Landau, G.M. and Guy, G. (1991), "Parallel (PRAM EREW) Algorithms for Contour-Based 2D Shape Recognition," *Pattern Recognition*, vol. 24, no. 109, pp. 929-942.

[Dor86] Dorst, L. (1986), "Pseudo-Euclidean Skeletons," *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 286-288, IEEE, Paris, France.

[Dos77] Doster, W. (1977), "Contextual Postprocessing System for Cooperation with a Multiple Choice Character Recognition System," *IEEE Transactions on Computers*, vol. 26, pp. 1090-1101.

[Doy60] Doyle, W. (1960), "Recognition of Sloppy, Handprinted Characters," *Proceedings of the Western Joint Computer Conference*, vol. 17, pp. 133-142.

[DP81] Davis, E.R. and Plummer, A.P.N. (1981), "Thinning Algorithms: A Critique and a New Methodology," *Pattern Recognition*, vol. 14, no. 1, pp. 53-63.

[DT70] Donaldson, R.W. and Toussaint, G.T. (1970), "Use of Contextual Constraints in Recognition of Contour-Traced Handprinted Characters," *IEEE Transactions on*

Computers, vol. 19, pp. 1096-1099.

[DT91] Downton, A.C. and Tregidgo, R.W.S. (1991), "The Use of a Trie Structured Dictionary as a Contextual Aid to Recognition of Handwritten British Postal Addresses," *Proceedings of the 1st International Conference on Document Analysis and Recognition*, pp. 594-602, Saint-Malo, France.

[Dud70] Duda, R.O. (1970), "Elements of Pattern Recognition," in *Adaptive, Learning and Pattern Recognition Systems*, ed. J.M. Mendal and K.-S. Fu, Academic Press.

[Dut74] Dutta, A.K. (1974), "An Experimental Procedure for Handwritten Character Recognition," *IEEE Transactions on Computers*, vol. 23, pp. 536-545.

[EC90] Elliman, D.G. and Connor, P. (1990), "The Creation of Topological Outlines from Binary Images," *Proceedings of Vision Interface '90*, pp. 53-60., Canadian Image Processing and Pattern Recognition Society, Halifax, Nova Scotia, Canada.

[EL90] Elliman, D.G. and Lancaster, I.T. (1990), "A Review of Segmentation and Contextual Analysis Techniques for Text Recognition," *Pattern Recognition*, vol. 23, pp. 337-346.

[Ell93] Elliman, D.G. (1993), "Ideas for Improved Vectorization," Private communication.

[EUF90] Edelman, S., Ullman, S. and Flash, T. (1990), "Reading Cursive Script by Alignment of Letter Prototypes," *International Journal of Computer Vision*, vol. 5, no. 3, pp. 303-331.

[Eve93] Evett, L. (1993), "The Use of Linguistic Information in Script Recognition," British Machine Vision Association and Society for Pattern Recognition Technical Meeting on Handwritten Character and Script Recognition.

[Fah88] Fahlman, S.E. (1988), "Faster-Learning Variations on Back-Propagation: An Empirical Study," in *Proceedings of the 1988 Connectionist Models Summer School*, ed. D.S. Touretzky, G.E. Hinton and T.J. Sejnowski, pp. 38-51, Morgan Kaufmann Publishers, Los Altos, California, U.S.A.

[Fah91] Fahlman, S.E. (1991), "The Recurrent Cascade-Correlation Architecture," in *Advances in Neural Information Processing Systems 3*, ed. R.P. Lippmann, J.E. Moody and D.S. Touretzky, pp. 190-196, Morgan Kaufmann Publishers, Los Altos, California, U.S.A.

[Fan86] Fanty, M. (1986), "Context-Free Parsing with Connectionist Networks," in *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, pp. 140-145, American Institute of Physics, New York, U.S.A.

[Fan87] Fan, T.I. (1987), "Optimal Matching of Deformed Patterns with Positional Influence," *Information Science*, vol. 41, pp. 259-280.

[FB86] Fu, K.-S. and Booth, T.L. (1986), "Grammatical Inference: Introduction and Survey — Part 1," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 3, pp. 343-375.

[FCC67] Fu, K.-S., Chien, Y.T. and Cardillo, G.P. (1967), "A Dynamic Programming Approach to Sequential Pattern Recognition," *IEEE Transactions on Computers*, vol. 16, pp. 790-803.

[Fis36] Fisher, R.A. (1936), "The Use of Multiple Measurements in Taxonomic Problems," (reprinted in) *Contributions to Mathematical Statistics*, John Wiley and Sons, New York, U.S.A., 1950.

[FK83] Favre, A. and Keller, H. (1983), "Parallel Syntactic Thinning by Recoding of Binary Pictures," *Computer Vision, Graphics and Image Processing*, vol. 23, pp. 99-112.

[FK88] Fletcher, L.A. and Kasturi, R. (1988), "A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6. pp. 910-918.

[FKH76] Fujimoto, Y., Kadota, S., Hayashi, S., Yamamoto, M., Yajima, S. and Yasuda, M. (1976), "Recognition of Handprinted Characters by Nonlinear Elastic Matching," *Proceedings of the 3rd International Joint Conference on Pattern Recognition*, pp. 113-118.

[FL90] Fahlman, S.E. and Lebiere, C. (1990), "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky, pp. 524-532, Morgan Kaufmann Publishers, Los Altos, California, U.S.A.

[For71] Forney, Jr, G.D. (1973), "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268-278.

[For90] Forsyth, R. (1990), "Neural Learning Algorithms: Some Experimental Trials," *Proceedings of the 3rd Conference on Neural Networks and Their Applications*, pp. 301-317, Nimes, France.

[FR88] Fromkin, V. and Rodman, R. (1988), *An Introduction to Language*, Holt, Rinehart and Winston, Inc., New York, U.S.A., 4th Ed.

[Fre60] Fredkin, E. (1960), "Trie Memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490-500.

[Fre61] Freeman, H. (1961), "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computers*, pp. 260-268.

[Fre74] Freeman, H. (1974), "Computer Processing of Line-Drawing Images," *Computational Surveys*, vol. 6, pp. 57-97.

[Fu68] Fu, K.-S. (1968), *Sequential Methods in Pattern Recognition and Machine Learning*, Academic Press.

[Fu80] Fu, K.-S. (1980), "Syntactic Image Modeling Using Stochastic Tree Grammars," *Computer Graphics and Image Processing*, vol. 12, pp. 136-152.

[Fu82] Fu, K.-S. (1982), *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.

[Fu86] Fu, K.-S. (1986), "Chapter 4 — Syntactic Pattern Recognition," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 85-117, Academic Press.

[FU94] Ferreira, A. and Ubéda, S. (1994), "Ultra-Fast Parallel Contour Tracking with Applications to Thinning," *Pattern Recognition*, vol. 27, no. 7, pp. 867-878.

[Fuk72] Fukunaga, K. (1972), *Introduction to Statistical Pattern Recognition*, Academic Press.

[Fuk75] Fukushima, K. (1975), "Cognitron: A Self-Organizing Multilayered Neural Network," *Biological Cybernetics*, vol. 20, pp. 121-136.

[Fuk80] Fukushima, K. (1980), "Neocognitron: A Self-Organizing Neural Network for a Mechanism of Pattern Recognition Unaffected by a Shift in Position," *Biological Cybernetics*, vol. 36, pp. 193-202.

[Fuk86a] Fukunaga, K. (1986), "Chapter 1 — Statistical Pattern Classification," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 3-32, Academic Press.

[Fuk86b] Fukushima, K. (1986), "A Neural Network Model for Selective Attention in Visual Pattern Recognition," *Biological Cybernetics*, vol. 55, pp. 5-15.

[Fuk88] Fukushima, K. (1988), "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition," *Neural Networks*, vol. 1, no. 2, pp. 119-130.

[Fuk89] Fukushima, K. (1989), "Analysis of the Process of Visual Pattern Recognition by the Neocognitron," *Neural Networks*, vol. 2, no. 6, pp. 413-420.

[Gal88] Gallant, S.I. (1988), "Connectionist Expert Systems," *Communications of the ACM*, vol. 31, no. 2, pp. 152-169.

[GE88] Goshtasby, A. and Ehrich, R.W. (1988), "Contextual Word Recognition Using Probabilistic Relaxation Labeling," *Pattern Recognition*, vol. 21, no. 5, pp. 455-462.

[GG89] Gath, I. and Geva, A. (1989), "Unsupervised Optimal Fuzzy Clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 773-781.

[GL92] Gan, K.W. and Lua, K.T. (1992), "Chinese Character Classification Using an Adaptive Resonance Network," *Pattern Recognition*, vol. 25, no. 8, pp. 877-882.

[Glu71] Glucksman, H.A. (1971), "Multicategory Classification of Patterns Represented by High-Order Vectors of Multilevel Measurements," *IEEE Transactions on Computers*, vol. 20, pp. 1593-1598.

[GMW68] Genchi, H., Mori, K.I., Watanabe, S. and Katsuragi, S. (1968), "Recognition of Handwritten Numeral Characters for Automatic Letter Sorting," *Proceedings of the IEEE*, vol. 56, pp. 1292-1301.

[GN91] Green, A.D.P. and Noakes, P.D. (1991), "Solving the Interconnection Problem with a Linked Assembly of Neural Networks," *IEE Proceedings - F Radar And Signal Processing*, vol. 138, no. 1, pp. 63-72.

[GR82] Gersho, A. and Ramamurthi, B. (1982), "Image Coding Using Vector Quantization," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 428-431.

[Gra72] Granlund, G.H. (1972), "Fourier Preprocessing for Hand Print Character Recognition," *IEEE Transactions on Computers*, vol. 21, pp. 195-210.

[Gra84] Gray, R.M. (1984), "Vector Quantization," *IEEE Acoustics, Speech and Signal Processing Magazine*, vol. 1, pp. 4-29.

[Gre60] Greville, T.N.E. (1960), "Some Applications of the Pseudoinverse of a Matrix," *Society for Industrial and Applied Mathematics Review*, vol. 2, no. 1, pp. 15-22.

[Gro68] Grossberg, S. (1968), "Some Nonlinear Networks Capable of Learning a Spatial Pattern of Arbitrary Complexity," *Proceedings of the National Academy of Sciences*, vol. 59, pp. 368-372.

[Gro72] Grossberg, S. (1972), "Neural Expectation: Cerebellar and Retinal Analogues of Cells Fired by Unlearnable and Learnable Pattern Classes," *Kybernetik*, vol. 10, pp. 49-57.

[Gro73] Grossberg, S. (1973), "Contour Enhancement, Short-Term Memory, and Constancies in Reverberating Networks," *Studies in Applied Mathematics*, vol. 52, pp. 217-257.

[Gro76a] Grossberg, S. (1976), "Adaptive Pattern Classification and Universal Recording: I. Parallel Development and Coding of Neural Detectors," *Biological Cybernetics*, vol. 23, pp. 121-134.

[Gro76b] Grossberg, S. (1976), "Adaptive Pattern Classification and Universal Recording: II. Feedback, Expectation, Olfaction, and Illusions," *Biological Cybernetics*, vol. 23, pp. 187-202.

[Gro82] Grossberg, S. (1982), *Studies of Mind and Brain: Neural Principals of Learning, Perception, Cognition, and Motor Control*, Reidel Press, Boston, U.S.A.

[Gro87] Grossberg, S. (1987), "Competitive Learning: From Interactive Activation to Adaptive Resonance," *Cognitive Science*, vol. 11, pp. 23-63.

[GS87] Gallant, S.I. and Smith, D. (1987), "Random Cells: An Idea Whose Time Has Come and Gone ... and Come Again?," *Proceedings of the 1st IEEE International Conference on Neural Networks*, vol. II, pp. 671-678, IEEE, San Diego, U.S.A.

[GS90] Govindan, V.K. and Shivaprasad, A.P. (1990), "Character Recognition — A Review," *Pattern Recognition*, vol. 23, no. 7, pp. 671-683.

[GT78] Gonzalez, R.C. and Thomason, M.G. (1978), *Syntactic Pattern Recognition*, Addison-Wesley, Reading, Mass., U.S.A.

[Güd76] Güdesen, A. (1976), "Quantitative Analysis of Preprocessing Techniques for the Recognition of Handprinted Characters," *Pattern Recognition*, vol. 8, pp. 219-227.

[GW77] Gonzalez, R.C. and Wintz, P.A. (1977), *Digital Image Processing*, Addison-Wesley, Reading, Mass., U.S.A.

[GW92] Garris, M.D. and Wilkinson, R.A. (1992), "NIST Special Database 3: Handwritten Segmented Characters," Image Recognition Group, Advanced Systems Division, National Institute of Standards and Technology.

[Har68] Hart, P.E. (1968), "The Condensed Nearest Neighbour Rule," *IEEE Transactions on Information Theory*, vol. 14, pp. 515-516.

[Har75] Hartigan, J.A. (1975), *Clustering Algorithms*, John Wiley and Sons, New York, U.S.A.

[Har84] Haralick, R.M. (1984), "Digital Step Edge from Zero Crossing of Second Directional Derivatives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, no. 1, pp. 58-68.

[HAS84] Hinton, G., Ackley, D. and Sejnowski, T. (1984), "Boltzmann Machines: Constraint Satisfaction Networks that Learn," Carnegie-Mellon University, Department of Computer Science, Technical Report CMU-CS-84-119.

[HB88] Hanson, S. and Burr, D. (1988), "Minkowski-r Back-Propagation: Learning in Connectionist Models with Non-Euclidean Error Signals," *Proceedings of the 1987 IEEE Conference on Neural Information Processing Systems — Natural and Synthetic*, pp. 348-357, American Institute of Physics, New York, U.S.A.

[HC86] Huang, J.S. and Chuang, K. (1986), "Heuristic Approach to Handwritten Numeral Recognition," *Pattern Recognition*, vol. 19, no. 1, pp. 15-19.

[HC87] Huang, J.S. and Chung, M.-L. (1987), "Separating Similar Complex Chinese Characters by Walsh Transform," *Pattern Recognition*, vol. 20, no. 4, pp. 425-428.

[HD80] Hall, P.A.V. and Dowling, G.R. (1980), "Approximate String Matching," *Computing Surveys*, vol. 12, no. 4, pp. 381-402.

[HD86] Ho, S.-B. and Dyer, C.R. (1986), "Shape Smoothing Using Medial Axis Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 512-520.

[Heb49] Hebb, D.O. (1949), "Chapter 4 — The First Stage of Perception: Growth of the Assembly," in *The Organisation of Behaviour*, John Wiley and Sons, New York, U.S.A.

[Hec82] Hecht-Nielson, R. (1982), "Neural Analog Processing," *Proceedings of the SPIE*, vol. 360, pp. 180-189.

[Hec87a] Hecht-Nielson, R. (1987), "Neural Network Nearest Matched-Filter Classification of Spatiotemporal Patterns," *Applied Optics*, vol. 26, pp. 1892-1899.

[Hec87b] Hecht-Nielson, R. (1987), "Counterpropagation Networks," *Applied Optics*, vol. 26, pp. 4979-4985.

[HF92] Hoehfeld, M. and Fahlman, S.E. (1992), "Learning with Limited Numerical Precision Using the Cascade-Correlation Learning Algorithm," *IEEE Transactions on Neural Networks*, vol. 3, no. 4, pp. 602-611.

[HHS90] Ho, T.K., Hull, J.J. and Srihari, S.N. (1990), "Combination of Structural Classifiers," *IAPR (International Association for Pattern Recognition) Workshop on Syntactic and Structural Pattern Recognition*, pp. 123-136, Murray Hill, New Jersey, U.S.A.

[Hig61] Highleyman, W.H. (1961), "An Analog Method for Character Recognition," *IRE Transactions on Electronic Computers*, vol. 10, pp. 502-512.

[Hig62] Highleyman, W.H. (1962), "Linear Decision Functions, with Application to Pattern Recognition," *Proceedings of the IRE*, vol. 50, pp. 1501-1514.

[Hil69] Hilditch, C.J. (1969), "Linear Skeletons from Square Cupboards," in *Machine Intelligence IV*, ed. B. Meltzer and D. Michie, pp. 403-420, University Press, Edinburgh, Scotland.

[Hil83] Hilditch, C.J. (1983), "Comparison of Thinning Algorithms on a Parallel Processor," *Image and Vision Computing*, vol. 1, no. 3, pp. 115-132.

[HK65] Ho, Y.-C. and Kashyap, R.L. (1965), "An Algorithm for Linear Inequalities and Its Applications," *IRE Transactions on Electronic Computers*, vol. 14, pp. 683-688.

[HK66] Ho, Y.-C. and Kashyap, R.L. (1966), "A Class of Iterative Procedures for Linear Inequalities," *SIAM Journal of Control*, vol. 4, pp. 112-115.

[HLN80] Hu, C.-H., Li, P., Ning, H.-Y. and Wu, F.-F. (1980), "A Handwritten Numeral Recognition Machine for Automatic Mail-Sorting,(rq in *EUSIPCO-80 Signal Processing: Theories and Applications*, ed. M. Kunt and F. de Coulon, North-Holland, Amsterdam.

[Hoa62] Hoare, C.A.R. (1962), "Quicksort," *Computer Journal*, vol. 5, pp. 10-15.

[Hop82] Hopfield, J.J. (1982), "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, vol. 79, pp. 2554-2558.

[Hos72] Hosking, K.H. (1972), "A Contour Method for the Recognition of Handprinted Characters," in *Machine Perception of Patterns and Pictures*, The Institute of Physics, London, England.

[Hou62] Hough, P.V.C (1962), "A Method and Means for Recognizing Complex Patterns," U.S. Patent 3069654.

[HRF76] Hanson, A.R., Riseman, E.M. and Fisher, E. (1976), "Context in Word Recognition," *Pattern Recognition*, vol. 8, pp. 35-45.

[HS77] Hunt, J.E. and Szymanski, T.G. (1977), "Fast Algorithm for Computing Longest Common Subsequences," *Communications of the ACM*, vol. 20, pp. 350-353.

[HS82] Hull, J.J. and Srihari, S.N. (1982), "Experiments in Text Recognition with Binary *N*-Gram and Viterbi Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, no. 5, pp. 520-530.

[HS86] Hull, J.J. and Srihari, S.N. (1986), "A Computational Approach to Word Shape Recognition: Hypothesis Generation and Testing," *Proceedings of the IEEE-CS Conference on Computer Vision and Pattern Recognition*, pp. 156-161.

[HS88] Hertz, L. and Schafer, R.W. (1988), "Multilevel Thresholding Using Edge Matching," *Computer Vision, Graphics and Image Processing*, vol. 44, pp. 279-295.

[HSC83] Hull, J.J., Srihari, S.N. and Choudhari, R. (1983), "An Integrated Algorithm for Text Recognition: Comparison with a Cascaded Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 4, pp. 384-395.

[HSC87] Holt, C.M., Stewart, A., Clint, M. and Perrott, R.H. (1987), "An Improved Parallel Thinning Algorithm," *Communications of the ACM*, vol. 30, no. 2, pp. 156-160.

[HSC88] Hull, J.J., Srihari, S.N., Cohen, E., Kaan, L., Cullen, P. and Palumbo, P. (1988), "A Blackboard-Based Approach to Handwritten ZIP Code Recognition," *Proceedings of the 9th International Conference on Pattern Recognition*, vol. 1, pp. 111-113, IEEE, Rome, Italy.

[HT85] Hopfield, J.J. and Tank, D.W. (1986), "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, pp. 141-152.

[HT86] Hopfield, J.J. and Tank, D.W. (1986), "Computing with Neural Circuits: A Model," *Science*, vol. 233, pp. 625-633.

[HTD72] Hussain, A.B.S., Toussaint, G.T. and Donaldson, R.W. (1972), "Results Obtained Using a Simple Character Recognition Procedure on Munson's

Handprinted Data," *IEEE Transactions on Computers*, vol. 21, pp. 201-205.

[Hu62] Hu, M.K. (1962), "Visual Pattern Recognition by Moment Invariants," *IRE Transactions on Information Theory*, vol. 8, pp. 179-187.

[HU79] Hopcroft, J.E. and Ullman, J.D. (1979), *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, Mass., U.S.A.

[Hue73] Hueckel, M.H. (1973), "A Local Visual Edge Operator Which Recognizes Edges and Lines," *Journal of the ACM*, vol. 20, pp. 634-647.

[Hul94] Hull, J.J. (1994), "A Database for Handwritten Text Recognition Research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550-554.

[HZ83] Hummel, R.A. and Zucker, S.W. (1983), "On the Foundations of Relaxation Labeling Processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 3, pp. 267-287.

[IK88] Illingworth, J. and Kittler, J. (1988), "A Survey of the Hough Transform," *Computer Vision, Graphics and Image Processing*, vol. 44, pp. 87-116.

[Jai86] Jain, A.K. (1986), "Chapter 2 — Cluster Analysis," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 33-57, Academic Press.

[Jam85] James, M. (1985), *Classification Algorithms*, Collins.

[JHL66] Johnson, D.D., Haugh, C.F. and Li, K.P. (1966), "The Application of a Few Hyperplane Decision Techniques to Handprinted Character Recognition," *Proceedings of the National Electronics Conference*, vol. 22, pp. 869-874.

[KAM83] Kushnir, M., Abe, K. and Matsumoto, K. (1983), "An Application of the Hough Transform to the Recognition of Printed Hebrew Characters," *Pattern Recognition*, vol. 16, no. 2, pp. 183-191.

[KAM85] Kushnir, M., Abe, K. and Matsumoto, K. (1985), "Recognition of Handprinted Hebrew Characters Using Features Selected in the Hough Transform Space," *Pattern Recognition*, vol. 18, pp. 103-114.

[Kan82] Kandel, A. (1982), *Fuzzy Techniques in Pattern Recognition*, John Wiley and Sons, New York, U.S.A.

[KB73] Krause, P. and Bleichrodt, H. (1973), "Experiments on Direct Input and Recognition of Handwritten Digits and Hand-Printed Letters with Computers," translated from *Mitteilungen aus dem Institut fur Informations-Verarbeitung in Technik and Biologie*, pp. 9-16, Karlsruhe, West Germany.

[KC94] Kopec, G.E. and Chou, P.A. (1994), "Document Image Decoding Using Markov Source Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 602-617.

[KD] Kanal, L.N. and Dattatreya, G.R., "Pattern Recognition," pp. 720-729.

[KEB91] Kiryati, N., Eldar, Y. and Bruckstein, A.M. (1991), "A Probabilistic Hough Transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303-316.

[KGV83] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983), "Optimisation by Simulated Annealing," *Science*, vol. 220, pp. 671-680.

[KI85] Kittler, J. and Illingworth, J. (1985), "A Review of Relaxation Labelling Algorithms," *Image and Vision Computing*, vol. 3, no. 4, pp. 206-216.

[Kit80] Kittler, J. (1980), "Automatic Interpretation of Symbolic Data on Maps and Engineering Drawings," *International Congress on Applied Systems Research and Cybernetics*, pp. 2400-2404.

[Kit86] Kittler, J. (1986), "Chapter 3 — Feature Selection and Extraction," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 59-83, Academic Press.

[KK87] Klein, F. and Kubler, O. (1987), "Euclidean Distance Transformations and Model-Guided Image Interpretation," *Pattern Recognition Letters*, vol. 5, no. 1, pp. 19-30.

[KKL92] Kohonen, T., Kangas, J., Laaksonen, J. and Torkkola, K. (1992), "LVQ_PAK: A Program Package for the Correct Application of Learning Vector Quantization Algorithms," *Proceedings of the International Joint Conference on Neural Networks*, vol I, pp. 725-730, IEEE, Baltimore, U.S.A.

[KL76] Kwon, S.K. and Lai, D.C. (1976), "Recognition Experiments with Handprinted Numerals," *Proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, pp. 74-83, Hyannis, Mass., U.S.A.

[KL86] Kay, S.M. and Lemay, G.J. (1986), "Edge Detection Using the Linear Method," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, pp. 1221-1227.

[KL88] Khotanzad, A. and Lu, J. (1988), "Distortion Invariant Character Recognition by a Multi-Layer Perceptron and Backpropagation Learning," *Proceedings of the 1st IEEE International Conference on Neural Networks*, vol. I, pp. 625-632, IEEE, San Diego, U.S.A.

[KLK88] Kim, S-D., Lee, J-H. and Kim, J-K. (1988), "A New Chain-Code Algorithm for Binary Images Using Run-Length Codes," *Computer Vision, Graphics and Image Processing*, vol. 41, pp. 114-128, Academic Press.

[Klo86] Klopf, A. (1986), "Drive-Reinforcement Model of Single Neuron Function: An Alternative to the Hebbian Neuronal Model," in *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, pp. 265-270, American Institute of Physics, New York, U.S.A.

[KLS89] Krzyzak, A., Leung, S.Y. and Suen, C.Y. (1989), "Reconstruction of Two-Dimensional Patterns from Fourier Descriptors," *Machine Vision and Applications Journal*, vol. 2, pp. 123-140.

[Kno69] Knoll, A.L. (1969), "Experiments with 'Characteristic Loci' for Recognition of Handprinted Characters," *IEEE Transactions on Computers*, vol. 18, pp. 366-372.

[Knu68] Knuth, D.E. (1968), "Semantics of Context-Free Languages," *Mathematical Systems Theory*, vol. 2, no. 2, pp. 127-146.

[Knu73] Knuth, D.E. (1973), "Digital Searching," in *The Art of Computer Programming, Vol. 3*, pp. 481-499, Addison-Wesley, Reading, Mass., U.S.A.

[KO84] Kashyap, R.L. and Oommen, B.J. (1984), "Spelling Correction Using Probabilistic Methods," *Pattern Recognition Letters*, vol. 2, no. 3, pp. 147-154.

[Koh82] Kohonen, T. (1982), "Self-Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, vol. 43, pp. 59-69.

[Koh84] Kohonen, T. (1984), *Self-Organisation and Associative Memory*, Springer-Verlag, *Springer Series in Information Sciences*, vol. 8, 3rd Ed., 1989.

[Koh86] Kohonen, T. (1986), "Learning Vector Quantization for Pattern Recognition," Helsinki University of Technology, Department of Technical Physics, Technical Report TKK-F-A-601.

[Koh90a] Kohonen, T. (1990), "Improved Versions of Learning Vector Quantization," *Proceedings of the International Joint Conference on Neural Networks*, vol. I, pp. 545-550, IEEE, San Diego, U.S.A.

[Koh90b] Kohonen, T. (1990), "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480.

[Koh90c] Kohonen, T. (1990), "Statistical Pattern Recognition Revisited," in *Advanced Neural Computers*, pp. 137-144.

[Kos86] Kosko, B. (1986), "Differential Hebbian Learning," in *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, pp. 277-282, American Institute of Physics, New York, U.S.A.

[Kos87] Kosko, B. (1987), "Adaptive Bidirectional Associative Memories," *Applied Optics*, vol. 26, pp. 4947-4960.

[Kos88] Kosko, B. (1988), "Bidirectional Associative Memories," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 42-60.

[Koz71] Kozlay, D. (1971), "Feature Extraction in an Optical Character Recognition Machine," *IEEE Transactions on Computers*, vol. 20, pp. 1063-1067.

[KPB87] Kahan, S., Pavlidis, T. and Baird, H.S. (1987), "On the Recognition of Printed Characters of any Font and Size," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 2, pp. 274-287.

[KPS79] Kwan, C.C., Pang, L. and Suen, C.Y. (1979), "A Comparative Study of Some Recognition Algorithms in Character Recognition", *Proceedings of the International Conference on Cybernetics and Society*, pp. 530-535, IEEE, Tucson, Arizona, U.S.A.

[KR73] Kohonen, T. and Ruohonen, M. (1973), "Representation of Associated Data by Matrix Operators," *IEEE Transactions on Computers*, vol. 22, pp. 701-702.

[KR78] Kohonen, T. and Reuhkala, E. (1976), "A Very Fast Associative Method for the Recognition and Correction of Misspelt Words, Based on Redundant Hash Addressing," *Proceedings of the 4th International Joint Conference on Pattern Recognition*, pp. 807-809, Kyoto, Japan.

[KS91] Kimura, F. and Shridhar, M. (1991), "Recognition of Connected Numeral Strings," *Proceedings of the 1st International Conference on Document Analysis and Recognition*, Saint-Malo, France.

[KS92] Kimura, F. and Shridhar, M. (1991), "Segmentation-Recognition Algorithm for ZIP Code Field Recognition," *Machine Vision and Applications*, vol. 5, no. 3, pp. 199-210.

[KSP74] Krause, P., Schwerdtman, W. and Paul, D. (1974), "Two Modifications of a Recognition System with a Pattern Series Expansion and Bayes Classifier," *Proceedings of the 2nd International Joint Conference on Pattern Recognition*, pp. 215-219.

[Kuh63] Kuhl, F. (1963), "Classification and Recognition of Hand-Printed Characters," *IEEE International Convention Record*, vol. 11, pp. 75-93.

[Kuk84] Kuklinski, T.T. (1984), "Components of Handprint Style Variability," *Proceedings of the 7th International Conference on Pattern Recognition*, pp. 924-926, IEEE, Montreal, Canada.

[Kun89] Kuner, P. (1989), "Matching of Attributed and Non-Attributed Graphs by Use of a Boltzmann Machine Algorithm," *Proceedings of the 1st IEE International Conference on Artificial Neural Networks*, pp. 369-373.

[Kwo88] Kwok, P.C.K. (1988), "A Thinning Algorithm by Contour Generation," *Communications of the ACM*, vol. 31, no. 11, pp. 1314-1324.

[LA87] Laarhoven, P.J.M. and Aarts, E.H.L. (1987), "Simulated Annealing: Theory and Applications," D. Reidel, Dordrecht, Holland.

[LB87] Leavers, V.F. and Boyce, J.F. (1986), "The Radon Transform and its Application to Shape Parameterization in Machine Vision," *Image and Vision Computing*, vol. 5, pp. 161-166.

[LBG80] Linde, Y., Buzo, A. and Gray, R.M. (1980), "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84-95.

[LC83] Li, H.F. and Cheng, S.C. (1983), "Projection Profile and Fourier Transform for Chinese Character Recognition," *International Journal of Electronics (UK)*, vol. 54, pp. 299-300.

[LCS84] Landy, M.S., Cohen, Y. and Sperling, G. (1984), "HIPS — A UNIX-Based Image-Processing System," *Computer Vision, Graphics and Image Processing*, vol. 25, no. 3, pp. 331-347.

[LD87] Leedham, C.G. and Downton, A.C. (1987), "Automatic Recognition and Transcription of Pitman's Handwriting Shorthand — An Approach to Short Forms," *Pattern Recognition*, vol. 20, pp. 341-348.

[Lea90] Leavers, V.F. (1990), "Use of the Radon Transform for Robust Feature Extraction in Optical Character Recognition," *IAPR (International Association for Pattern Recognition) Workshop on Syntactic and Structural Pattern Recognition*, pp. 249-255, Murray Hill, New Jersey, U.S.A.

[Lev66] Levenshtein, V.I. (1966), "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics-Doklady*, vol. 10, no. 8, pp. 707-710, translated from *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, pp. 845-848, 1965.

[Lev81] Levine, B. (1981), "Derivations of Tree Sets with Applications to Grammatical Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, no. 3, pp. 285-293.

[LF] Leedham, C.G. and Friday, P.D., "Isolating Individual Handwritten Characters," pp. 4/1-4/7.

[LG87] Lippman, R. and Gold, B. (1987), "Neural Net Classifiers Useful for Speech Recognition," *Proceedings of the 1st IEEE International Conference on Neural Networks*, vol. IV, pp. 417-426, IEEE, San Diego, U.S.A.

[Lia81] Liao, Y.-Z. (1981), "A Two-Stage Method of Fitting Conic Arcs and Straight-Line Segments to Digitized Contours," *Proceedings of the International Conference on Pattern Recognition and Image Processing*, pp. 224-229, IEEE, Dallas, Texas, U.S.A.

[Lip87] Lippman, R.P. (1987), "An Introduction to Computing with Neural Nets," *IEEE Acoustic Speech and Signal Processings Magazine*, vol. 4, pp. 4-22.

[LLS92] Lam, L., Lee, S.-W. and Suen, C.Y. (1992), "Thinning Methodologies — A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 869-885.

[Low91] Low, A. (1991), *Introductory Computer Vision and Image Processing*, McGraw-Hill Book Company (UK) Limited.

[LRS91] Lu, S.W., Ren, Y. and Suen, C.Y. (1991), "Hierarchical Attributed Graph Representation and Recognition of Handwritten Chinese Characters," *Pattern Recognition*, vol. 24, no. 7, pp. 617-632.

[LS81] Lai, M.T.Y. and Suen, C.Y. (1981), "Automatic Recognition of Characters by Fourier Descriptors and Boundary Line Encodings," *Pattern Recognition*, vol. 14, pp. 383-393.

[LS88a] Lam, L. and Suen, C.Y. (1988), "Structural Classification and Relaxation Matching of Totally Unconstrained Handwritten ZIP Code Numbers," *Pattern Recognition*, vol. 21, no. 1, pp. 19-31.

[LS88b] Leavers, V.F. and Sandler, M.B. (1988), "An Efficient Radon Transform," in *Lecture Notes in Computer Science*, vol. 301, *Proceedings of the British Pattern Recognition Association 4th International Conference on Pattern Recognition*, ed. J. Kittler, pp. 380-389, Springer-Verlag.

[LSA94] Liang, S., Shridhar, M. and Ahmadi, M. (1994), "Segmentation of Touching Characters in Printed Document Recognition," *Pattern Recognition*, vol. 27, no. 6, pp. 825-840.

[LT93] Lanitis, A. and Taylor, C. (1993), "Reading Handwritten Postcodes Using Flexible Template Matching," British Machine Vision Association and Society for Pattern Recognition Technical Meeting on Handwritten Character and Script Recognition.

[Lu79] Lu, S.T. (1979), "A Tree-to-Tree Distance and Its Application to Cluster Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 219-224.

[Lue84] Luenberger, D.G. (1984), "Linear and Nonlinear Programming," Addison-Wesley, Reading, Mass., U.S.A.

[LW86] Lü, H.E. and Wang, P.S.P. (1986), "A Comment on 'A Fast Parallel Thinning Algorithm for Digital Patterns,'" *Communications of the ACM*, vol. 29, no. 3, pp. 239-242.

[MAC92] Musavi, M.T., Ahmed, W., Chan, K.H., Faris, K.B. and Hummels, D.M. (1992), "On the Training of Radial Basis Function Classifiers," *Neural Networks*, vol. 5, no. 4, pp. 595-603.

[Mae90] Maes, M. (1990), "On a Cyclic String-to-String Correction Problem," *Information Processing Letters*, vol. 35, pp. 73-78.

[Mae91] Maes, M. (1991), "Polygonal Shape Recognition Using String-Matching Techniques," *Pattern Recognition*, vol. 24, no. 5, pp. 433-440.

[Mah36] Mahalanobis, P.C. (1936), "On the Generalized Distance in Statistics," *Proceedings of the National Institute of Science (India)*, vol. 12, pp. 49-55.

[Mal73] Malsburg, C. v.d. (1973), "Self-Organization of Orientation Sensitive Cells in the Striate Cortex," *Kybernetik*, vol. 14, pp. 85-100.

[Man83] Manohar, M. (1983), "Edge Detection via Optimal Clustering," *Proceedings of the International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 1076-1078, Halifax, Canada.

[Man86] Mantas, J. (1986), "An Overview of Character Recognition Methodologies," *Pattern Recognition*, vol. 19, no. 6, pp. 425-430.

[Mar89] Marshall, S. (1989), "Review of Shape Coding Techniques," *Image and Vision Computing*, vol. 7, no. 4, pp. 281-294.

[Mat55] Matusita, K. (1955), "Decision Rules Based on the Distance for Problems of Fit, Two Samples and Estimation," *Annals of Mathematical Statistics*, vol. 26, pp. 631-640.

[MD89] Moody, J. and Darken, C.J. (1989), "Fast Learning in Networks of Locally Tuned Processing Units," *Neural Computation*, vol. 1, pp. 281-294.

[Mey88] Meyer, F. (1988), "Skeletons in Digital Spaces," in *Image Analysis and Mathematical Morphology; Volume 2: Theoretical Advances*, ed. J. Serra, ch. 13, pp. 258-296, Academic Press.

[MF75] Merlin, P.M. and Farber, D.J. (1975), "A Parallel Mechanism for Detecting Curves in Pictures," *IEEE Transactions on Computers*, vol. 24, pp. 96-98.

[MG60] Marill, T. and Green, D.M. (1960), "Statistical Recognition Functions and the Design of Pattern Recognizers," *IRE Transactions on Electronic Computers*, vol. 9, pp. 472-477.

[MG89] Mitchell, B.T. and Gillies, A.M. (1989), "A Model-Based Computer Vision System for Recognizing Handwritten ZIP Codes," *Machine Vision and Applications*, vol. 2, pp. 231-243.

[MG94] Man, Y. and Gath, I. (1994), "Detection and Separation of Ring-Shaped Clusters Using Fuzzy Clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 8, pp. 855-861.

[MGK90] Miller, W.T., Glanz, F.H. and Kraft, L.G. (1990), "CMAC: An Associative Neural Network Alternative to Backpropagation," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1561-1567.

[MH62] Masterson, J.L. and Hirsch, R.S. (1962), "Machine Recognition of Constrained Handwritten Arabic Numerals," *IRE Transactions on Human Factors in Electronics*, vol. 3, pp. 62-65.

[MH80] Marr, D. and Hildreth, E. (1980), "Theory of Edge Detection," *Proceedings of the Royal Society of London*, vol. B207, pp. 187-217.

[MH83] Mantas, J. and Heaton, A.G. (1983), "Handwritten Character Recognition by Parallel Labelling and Shape Analysis," *Pattern Recognition Letters*, vol. 1, pp. 465-468.

[Mic86] Miclet, L. (1986), *Structural Methods in Pattern Recognition*, Springer-Verlag, New York, U.S.A.

[MJN87] Martínez-Pérez, M.P., Jiménez, J. and Navalón, J. (1987), "A Thinning Algorithm Based on Contours," *Computer Vision, Graphics and Image Processing*, vol. 39, pp. 186-201.

[MMY75] Mori, S., Mori, T., Yamanoto, K., Yamada, H., Saito, T. and Nakata, K. (1975), "Recognition of Handprinted Characters," *Proceedings of the 2nd USA-Japan Computer Conference*, pp. 18-23.

[Mon68] Montanari, U. (1968), "A Method for Obtaining Skeletons Using a Quasi-Euclidean Distance," *Journal of the ACM*, vol. 15, pp. 600-624.

[MP43] McCulloch, W.S. and Pitts, W. (1943), "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 9, pp. 127-147.

[MP69a] Miloslavskaya, N.K. and Polyakov, S.T. (1969), "Minsk-I Digital Computer Simulation of an Algorithm for Handwritten Character Recognition," *Soviet Automatic Control*, vol. 14, pp. 42-45.

[MP69b] Minsky, M. and Papert, S. (1969), *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Mass., U.S.A.

[MRG85] Makhoul, J., Roucos, S. and Gish, H. (1985), "Vector Quantization in Speech Coding," *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551-1588.

[MS54] Motzkin, T.S. and Schoenberg, I.J. (1954), "The Relaxation Method for Linear Inequalities," *Canadian Journal of Mathematics*, vol. 6, pp. 393-404.

[MST94] Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (1994), *Machine Learning: Neural and Statistical Classification*, Ellis Horwood.

[MT77] Muth, F.E. and Tharp, A.L. (1977), "Correcting Human Error in Alphanumeric Terminal Input," *Information Processing and Management*, vol. 13, pp. 329-337.

[Mun68] Munson, J.H. (1968), "Experiments in the Recognition of Hand-Printed Text: Part I — Character Recognition," *Proceedings of the Fall Joint Computer Conference*, vol. 33, pp. 1125-1136.

[MV93] Marzal, A. and Vidal, E. (1993), "Computation of Normalized Edit Distance and Applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 926-932.

[Nay71] Naylor, W.C. (1971), "Some Studies in the Interactive Design of Character Recognition Systems," *IEEE Transactions on Computers*, vol. 20, pp. 1075-1086.

[NB80] Nevatia, R. and Babu, K.R. (1980), "Linear Feature Extraction and Description," *Computer Graphics Image Processing*, vol. 13, pp. 257-269.

[NDG80] Ni, G., Ding, J., Gao, Z. and Liu, J. (1980), "A Structural Method for Hand-printed Alphanumeric and Other Symbols," *Proceedings of the 5th International Conference on Pattern Recognition*, pp. 726-728, IEEE.

[Neu75] Neuhoff, D.L. (1975), "The Viterbi Algorithm as an Aid in Text Recognition," *IEEE Transactions on Information Theory*, vol. 21, pp. 222-226.

[Nev86] Nevatia, R. (1986), "Chapter 9 — Image Segmentation," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 215-231, Academic Press.

[NF77] Narendra, P.M. and Fukunaga, K. (1977), "A Branch and Bound Algorithm for Feature Subset Selection," *IEEE Transactions on Computers*, vol. 26, no. 9, pp. 917-922.

[Nie77] Niemann, H. (1977), "Classification of Characters by Man and by Machine," *Pattern Recognition*, vol. 9, pp. 173-179.

[NS84] Naccache, N.J. and Shinghal, R. (1984), "SPTA: A Proposed Algorithm for Thinning Binary Patterns," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 14, no. 3, pp. 409-418.

[NT70] Nagy, G. and Tuong, N. (1970), "Normalization Techniques for Handprinted Numerals," *Communications of the ACM*, vol. 13, no. 8, pp. 475-481.

[OD88] Oulamara, A. and Duvernoy, J. (1988), "An Application of the Hough Transform to Automatic Recognition of Berber Characters," *Signal Processing (Netherlands)*, vol. 14, pp. 79-90.

[OGo90] O'Gorman, L. (1990), "$k \times k$ Thinning," *Computer Vision, Graphics and Image Processing*, vol. 51, pp. 195-215.

[OK91] O'Hair, M.A. and Kabrisky, M. (1991), "Recognizing Whole Words as Single Symbols," *Proceedings of the 1st International Conference on Document Analysis and Recognition*, pp. 350-358, Saint-Malo, France.

[Ota75] Ota, P.A. (1975), "Mosaic Grammars," *Pattern Recognition*, vol. 7, pp. 61-65.

[OTK76] Okuda, T., Tanaka, E. and Kasai, T. (1976), "A Method for the Correction of Garbled Words based on the Levenshtein Metric," *IEEE Transactions on Computers*, vol. 25, no. 2, pp. 172-178.

[Ott74] Ott, R. (1974), "On Feature Selection by Means of Principal Axis Transform and Nonlinear Classification," *Proceedings of the 2nd International Joint Conference on Pattern Recognition*, pp. 220-222.

[Pal79] Palm, G. (1979), "On Representation and Approximation of Nonlinear Systems, Part II: Discrete Time," *Biological Cybernetics*, vol. 34, pp. 49-52.

[Pav72] Pavlidis, T. (1972), "Linear and Context-Free Graph Grammars," *Journal of the ACM*, vol. 19, pp. 11-12.

[Pav76] Pavlidis, T. (1976), "The Use of Algorithms of Piecewise Approximations for Picture Processing Applications," *ACM Transactions on Mathematical Software*, vol. 2, pp. 305-321.

[Pav77] Pavlidis, T. (1977), *Structural Pattern Recognition*, Springer Publishing, New York, U.S.A.

[Pav78] Pavlidis, T. (1978), "A Minimum Storage Boundary Tracing Algorithm and Its Application to Automatic Inspection," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 8, no. 1, pp. 66-69.

[Pav80] Pavlidis, T. (1980), "A Thinning Algorithm for Discrete Binary Images," *Computer Graphics and Image Processing*, vol. 13, pp. 142-157.

[Pav82a] Pavlidis, T. (1982), *Algorithms for Graphics and Image Processing*, Computer Science Press, Rockville, MD, U.S.A.

[Pav82b] Pavlidis, T. (1982), "Curve Fitting as a Pattern Recognition Problem," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 853-858, IEEE, Munich, Germany.

[Pav82c] Pavlidis, T. (1982), "Vector and Arc Encoding of Graphics and Text," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 610-613, IEEE, Munich, Germany.

[Pav84] Pavlidis, T. (1984), "A Hybrid Vectorization Algorithm," *Proceedings of the 7th International Conference on Pattern Recognition*, pp. 490-492, IEEE, Montreal, Canada.

[Pav86] Pavlidis, T. (1986), "A Vectorizer and Feature Extractor for Document Recognition," *Computer Vision, Graphics and Image Processing*, vol. 35, pp. 111-127.

[Pen55] Penrose, R. (1955), "A Generalized Inverse for Matrices," *Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 406-413.

[Pen56] Penrose, R. (1956), "On Best Approximate Solutions of Linear Matrix Equations," *Proceedings of the Cambridge Philosophical Society*, vol. 52, pp. 17-19.

[Per80] Perkins, W.A. (1980), "Area Segmentation of Images Using Edge Points," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, pp. 8-15.

[PF69] Partick, E.A. and Fisher, F.P. (1969), "Nonparametric Feature Selection," *IEEE Transactions on Information Theory*, vol. 15, pp. 557-584.

[PF77] Persoon, E. and Fu, K.-S. (1977), "Shape Discrimination Using Fourier Descriptors," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 3, pp. 170-179.

[PF80] Pferd, W. and Stocker, G.C. (1980), "Optical Fibers for Scanning Digitizers," *The Bell System Technical Journal*, vol. 60, no. 4, pp. 523-533.

[PG87] Pérez, A. and Gonzalez, R.C. (1987), "An Iterative Thresholding Algorithm for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 6, pp. 742-751.

[PLH88] Pawlicki, T., Lee, D.-S., Hull, J. and Srihari, S. (1988), "Neural Networks and Their Application to Handwritten Digit Recognition," *Proceedings of the 1st IEEE International Conference on Neural Networks*, vol. II, pp. 63-70, IEEE, San Diego, U.S.A.

[PM89] Parks, P.C. and Militzer, J. (1989), "Convergence Properties of Associative Memory Storage for Learning Control Systems," *Proceedings of the IFAC Symposium on Adaptive Systems in Control and Signal Processing*, pp. 565-572, Pergamon Press, Oxford, England.

[POB87] Pizer, S.M., Oliver, W.R. and Bloomberg, S.H. (1987), "Hierarchical Shape Description via the Multiresolution Symmetric Axis Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 505-511.

[PR69] Pfaltz, J.L. and Rosenfeld, A. (1969), "Web Grammars," *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington, D.C., U.S.A.

[PR78] Panda, D.P. and Rosenfeld, A. (1978), "Image Segmentation by Pixel Classification in (Gray Level, Edge Value) Space," *IEEE Transactions on Computers*, vol. 27, pp. 875-879.

[Pri94] Priestnall, G. (1994), "Machine Recognition of Engineering Drawings," Ph.D. thesis, University of Nottingham, Nottingham, England.

[PS82] Pervez, A. and Suen, C.Y. (1982), "Segmentation of Unconstrained Handwritten Numeric Postal ZIP Codes," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 545-547, IEEE, Munich, Germany.

[QG92] Quint, D.J. and Gilmore, J.L. (1992), "Alexia Without Agraphia," *Neuroradiology*, vol. 34, no. 3, pp. 210-214.

[RAB89] Rapcsak, S.Z., Arthur, S.A., Bliklen, D.A. and Rubens, A.B. (1989), "Lexical Agraphia in Alzheimers-Disease," *Archives of Neurology*, vol. 46, no. 1, pp. 65-68.

[Rad17] Radon, J. (1917), "Uber die Bestimmung von Funktionen Durch Ihre Integralwerte Langs Gewisser Mannigfaltigkeiten," *Berichte Sachsische Akademie der Wissenschaften Leipzig, Math-Phys Kl.*, vol. 69, pp. 262-267.

[Ram72] Ramer, U. (1972), "An Iterative Procedure for Polygonal Approximation of Plane Curves," *Computer Graphics and Image Processing*, vol. 1, pp. 244-256.

[Rav67] Raviv, J. (1967), "Decision Making in Markov Chains Applied to the Problem of Pattern Recognition," *IEEE Transactions on Information Theory*, vol. 3, no. 4, pp. 536-551.

[RCC90] Riazanoff, S., Cervelle, B. and Chorowicz, J. (1990), "Parametrisable Skeletonization of Binary and Multi-Level Images," *Pattern Recognition Letters*, vol. 11, no. 1, pp. 25-33.

[RD76] Rosenfeld, A. and Davis, L.S. (1976), "A Note on Thinning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, pp. 226-228.

[RE71] Riseman, E.M. and Ehrich, R.W. (1971), "Contextual Word Recognition Using Binary Digrams," *IEEE Transactions on Computers*, vol. 20, no. 4, pp. 397-403.

[Rei91] Reiss, T.H. (1991), "The Revised Fundamental Theorem of Moment Invariants," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 830-834.

[RH74] Riseman, E.M. and Hanson, A.R. (1974), "A Contextual Postprocessing System for Error Correction Using Binary *N*-Grams," *IEEE Transactions on Computers*, vol. 23, no. 5, pp. 480-493.

[RHZ76] Rosenfeld, A., Hummel, R.A. and Zucker, S.W. (1976), "Scene Labelling by Relaxation Operations," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, pp. 420-443.

[Rip81] Ripley, B.D. (1981), *Spatial Statistics*, John Wiley and Sons, New York, U.S.A.

[RM86] Rumelhart, D.E. and McClelland, J.L. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, MIT Press, Cambridge, Mass., U.S.A.

[RMS89] Rajavelu, A., Musavi, M. and Shirvaikar, M. (1989), "A Neural Network Approach to Character Recognition," *Neural Networks*, vol. 2, no. 5, pp. 387-394.

[Ros58] Rosenblatt, F. (1958), "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, pp. 386-408.

[Ros62] Rosenblatt, R. (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington, DC, U.S.A.

[Ros73] Rosenfeld, A. (1973), "Array Grammar Normal Forms," *Information and Control*, vol. 23, no. 2, pp. 173-182.

[RP68] Rosenfeld, A. and Pfaltz, J.L. (1968), "Distance Functions on Digital Pictures," *Pattern Recognition*, vol. 1, pp. 33-61.

[Rut66] Rutovitz, D. (1966), "Pattern Recognition," *Journal of the Royal Statistics Society*, vol. 129, Series A, pp. 504-530.

[RZ85] Rumelhart, D.E. and Zipser, D. (1985), "Feature Discovery by Competetive Learning," *Cognitive Science*, vol. 9, pp. 75-112.

[SA87] Suzuki, S. and Abe, K. (1987), "Binary Picture Thinning by an Iterative Parallel Two-Subcycle Operation," *Pattern Recognition*, vol. 10, no. 3, pp. 297-307.

[Sar93] Sardana, H.K. (1993), *Edge Moments in Pattern Recognition*, Ph.D. thesis, University of Nottingham, Nottingham, England.

[Sar94] Sarle, W.S. (1994), "Neural Networks and Statistical Models," *Proceedings of the 19th Annual SAS Users Group International Conference*, pp. 1538-1550, SAS Institute, Cary, North Carolina, U.S.A.

[SB81] Sutton, R. and Barto, A. (1981), "Toward a Modern Theory of Adaptive Networks: Expectation and Prediction," *Psychological Review*, vol. 88, pp. 135-171.

[SB83] Srihari, S.N. and Bozinovic, R. (1983), "Use of Knowledge in the Visual Interpretation of Cursive Script," *Proceedings of the International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 187-191, Halifax, Canada.

[SB84] Shridhar, M. and Badreldin, A. (1984), "High Accuracy Character Recognition Algorithm Using Fourier and Topological Descriptors," *Pattern Recognition*, vol. 17, pp. 515-524.

[SB86] Shridhar, M. and Badreldin, A. (1986), "Recognition of Isolated and Simply Connected Handwritten Numerals," *Pattern Recognition*, vol. 19, no. 1, pp. 1-12.

[SB87] Shridhar, M. and Badreldin, A. (1986), "Context-Directed Segmentation Algorithm for Handwritten Numeral Strings," *Image and Vision Computing*, vol. 5, no. 1, pp. 3-9.

[SB91] Sarkar, S. and Boyer, K.L. (1991), "On Optimal Infinite Impulse Response Edge Detection Filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 11, pp. 1154-1171.

[SBM80] Suen, C.Y., Berthod, M. and Mori, S. (1980), "Automatic Recognition of Hand-printed Characters — The State of the Art," *Proceedings of the IEEE*, vol. 68, no. 4, pp. 469-487.

[SCH72] Sklansky, J., Chazin, R.L. and Hansen, B.J. (1972), "Minimum Perimeter Polygons of Digitized Silhouettes," *IEEE Transactions on Computers*, vol. 23, pp. 445-448.

[Sch89] Schalkoff, R.J. (1989), *Digital Image Processing and Computer Vision*, John Wiley and Sons, New York, U.S.A.

[Sch92] Schalkoff, R.J. (1992), *Pattern Recognition: Statistical, Structural and Neural Approaches*, John Wiley and Sons, New York, U.S.A.

[SDO86] Soffer, B., Dunning, G., Owechko, Y. and Marom, E. (1986), "Associative Holographic Memory with Feedback Using Phase-Conjugate Mirrors," *Optics Letters*, vol. 11, pp. 118-120.

[SE95] Sulaiman, M.N. and Evans, D.J. (1995), "Using a General-Purpose Neural-Network Simulation Tool — NEUCOMP — for Character-Recognition Problems," *Journal of Microcomputer Applications*, vol. 18, no. 1, pp. 65-81.

[Seb89] Sebesta, R.W. (1989), *Concepts of Programming Languages*, Benjamin Cummings, Redwood City, California, U.S.A.

[Sej77a] Sejnowski, T.J. (1977), "Storing Covariance with Nonlinearly Interacting Neurons," *Journal of Mathematical Biology*, vol. 64, pp. 303-321.

[Sej77b] Sejnowski, T.J. (1977), "Statistical Constraints on Synaptic Plasticity," *Journal of Mathematical Biology*, vol. 64, pp. 385-389.

[Sej88] Sejnowski, T.J. (1988), "Neural Network Learning Algorithms," in *Neural Computers*, ed. R. Eckmiller and C. v.d. Malsburg, NATO ASI Series, vol. F41, Springer-Verlag.

[SF83] Sanfeliu, A. and Fu, K.-S. (1983), "A Distance Measure Between Attributed Relational Graphs for Pattern Recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, no. 3, pp. 353-362.

[Sha70] Shaw, A.C. (1970), "Parsing of Graph-Representable Pictures," *Journal of the ACM*, vol. 17, no. 3, pp. 453-481.

[SHC83] Srihari, S.N., Hull, J.J. and Choudhari, R. (1983), "Integrating Diverse Knowledge Sources in Text Recognition," *ACM Transactions on Office Information Systems*, vol. 1, no. 1, pp. 68-87.

[She78] Sheil, B.A. (1978), "Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys," *Communications of the ACM*, vol. 21, no. 11, pp. 947-958.

[Shi73] Shimura, M. (1973), "Multicategory Learning Classifiers for Character Reading," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 3, pp. 74-85.

[Shi83] Shinghal, R. (1983), "A Hybrid Algorithm for Contextual Text Recognition," *Pattern Recognition*, vol. 16, no. 2, pp. 261-267.

[Shl88] Shlien, S. (1988), "Multifont Character Recognition for Typeset Documents," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, pp. 603-620.

[Shu94] Shustorovich, A. (1994), "Scale Specific and Robust Edge/Line Encoding with Linear Combinations of Gabor Wavelets," *Pattern Recognition*, vol. 27, no. 5, pp. 713-725.

[Sin84] Sinha, R.M.K. (1984), "Primitive Recognition and Skeletonization via Labeling," *Proceedings of the International Conference on Systems, Man and Cybernetics*, pp. 272-279, Halifax, Canada.

[Sin87a] Sinha, R.M.K. (1987), "A Width-Independent Algorithm for Character Skeleton Estimation," *Computer Vision, Graphics and Image Processing*, vol. 40, pp. 388-397.

[Sin87b] Sinha, R.M.K. (1987), "Some Characteristic Curves for Dictionary Organization with Digital Search," *IEEE Transactions on Systems, Man and Cybernetics*, vol.

17, no. 3, pp. 520-527.

[SJ92] Srinivasa, N. and Jouaneh, M. (1992), "A Neural Network Model for Invariant Pattern-Recognition," *IEEE Transactions on Signal Processing*, vol. 40, no. 6, pp. 1595-1599.

[SJ93] Srinivasa, N. and Jouaneh, M. (1993), "An Invariant Pattern-Recognition Machine Using a Modified ART Architecture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, pp. 1432-1437.

[SK86] Shlien, S. and Kubota, K. (1986), "Optical Character Recognition of Touching Characters," *Proceedings of Vision Interface '86*, pp. 390-395.

[SKH86] Sejnowski, T.J., Kienker, P.K. and Hinton, G.E. (1986), "Learning Symmetry Groups with Hidden Units: Beyond the Perceptron," *Physica*, vol. D22, pp. 260-275.

[Skl70] Sklansky, J. (1970), "Recognition of Convex Blobs," *Pattern Recognition*, vol. 2, pp. 3-10.

[SM83] Stentiford, F.W.M. and Mortimer, R.G. (1983), "Some New Heuristics for Thinning Binary Handprinted Characters for OCR," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, no. 1, pp. 81-84.

[SM92] Sabourin, M. and Mitiche, A. (1992), "Optical Character Recognition by a Neural Network," *Neural Networks*, vol. 5, no. 5, pp. 843-852.

[Smi68] Smith, F.W. (1967), "Pattern Classifier Design by Linear Programming," *IEEE Transactions on Computers*, vol. 17, pp. 367-372.

[Smi87] Smith, R.W. (1987), "Computer Processing of Line Images: A Survey," *Pattern Recognition*, vol. 20, no. 1, pp. 7-15.

[SMO86] Soffer, B., Marom, E., Owechko, Y. and Dunning, G. (1986), "Holographic Associative Memory Employing Phase Conjugation," *Proceedings of the SPIE*, vol. 684, pp. 2-6.

[SP88] Sinha, R.M.K., Prasada, B. (1988), "Visual Text Recognition Through Contextual Processing," *Pattern Recognition*, vol. 21, no. 5, pp. 463-479.

[Spa74] Spanjersberg, A.A. (1974), "Combinations of Different Systems for the Recognition of Handwritten Digits," *Proceedings of the 2nd International Joint Conference on Pattern Recognition*, pp. 208-209.

[SPH93] Sinha, R.M.K., Prasada, B., Houle, G.F. and Sabourin, M. (1993), "Hybrid Contextual Text Recognition with String-Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 915-925.

[SPS81] Shapiro, B., Pisa, J. and Sklansky, J. (1981), "Skeleton Generation from *x, y* Boundary Sequences," *Computer Graphics and Image Processing*, vol. 15, pp. 136-153.

[SR71] Stefanelli, R. and Rosenfeld, A. (1971), "Some Parallel Thinning Algorithms for Digital Pictures," *Communications of the ACM*, vol. 18, no. 2, pp. 255-264.

[SRT78] Shinghal, R., Rosenberg, D. and Toussaint, G.T. (1978), "A Simplified Heuristic Version of a Recursive Bayes Algorithm for Using Context in Text Recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 8, pp. 412-414.

[SS77] Suen, C.Y. and Shillman, R.J. (1977), "Low Error Rate Optical Character Recognition of Unconstrained Handprinted Letters Based on a Model of Human

Perception," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, pp. 491-495.

[SS83] Sarvarayudu, G.P.R. and Sethi, I.K. (1983), "Walsh Descriptors for Polygonal Curves," *Pattern Recognition*, vol. 16, pp. 327-336.

[SS88] Siedlecki, W. and Sklansky, J. (1988), "On Automatic Feature Selection," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 2, pp. 197-220.

[SSK77] Suen, C.Y., Shinghal, R. and Kwan, C.C. (1977), "Dispersion Factor: A Quantitative Measurement of the Quality of Handprinted Characters," *Proceedings of the International Conference on Cybernetics and Society*, pp. 681-685.

[SSS76] Suen, C.Y., Shiau, C., Shinghal, R. and Kwan, C.C. (1976), "Reliable Recognition of Handprint Data," *Proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, pp. 98-102, Hyannis, Mass., U.S.A.

[ST79a] Shinghal, R. and Toussaint, G.T. (1979), "Experiments in Text Recognition with the Modified Viterbi Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 184-193.

[ST79b] Shinghal, R. and Toussaint, G.T. (1979), "A Bottom-up and Top-Down Approach to Using Context in Text Recognition," *International Journal of Man-Machine Studies*, vol. 11, pp. 201-212.

[Ste61] Steinbuch, K. (1961), "Die Lernmatrix," *Kybernetik*, vol. 1, pp. 36-45.

[Ste91] Stephens, R.S. (1991), "Probabilistic Approach to the Hough Transform," *Image and Vision Computing*, vol. 9, pp. 66-71.

[Sto94] Stockley, P. (1994), "System Specification for Improving Match Quality," PAFEC Ltd Internal Report DTHC00M_SECT8.

[Str80] Stringa, L. (1980), "State of the Art and Perspectives of the Linguistic-Semantic Approach to Industrial OCR," in *Pattern Recognition in Practice*, ed. E.S. Gelsema and L.N. Kanal, pp. 325-335, North-Holland Publishing Company.

[Str90] Stringa, L. (1990), "A New Set of Constraint-Free Character Recognition Grammars," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 12, pp. 1210-1217.

[Sue82a] Suen, C.Y. (1982), " Distinctive Features in Automatic Recognition of Handprinted Characters," *Signal Processing*, vol. 4, pp. 193-207.

[Sue82b] Suen, C.Y. (1982), "The Role of Multi-Directional Loci and Clustering in Reliable Recognition of Characters," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 1020-1022, IEEE, Munich, Germany.

[Szu86] Szu, H. (1986), "Fast Simulated Annealing," in *AIP Conference Proceedings 151: Neural Networks for Computing*, ed. J. Denker, pp. 420-425, American Institute of Physics, New York, U.S.A.

[TA84] Tsuji, Y. and Asai, K. (1984), "Character Image Segmentation," *Proceedings of the Society of the Photo-Optical Institute of Engineers*, vol. 504, *Applications of Digital Image Processing VII*, pp. 2-9.

[TA91] Tsujimoto, S. and Asada, H. (1991), "Resolving Ambiguity in Resolving Touching Characters," *Proceedings of the 1st International Conference on Document Analysis and Recognition*, pp. 701-709, Saint-Malo, France.

[Tam78] Tamura, H. (1978), "A Comparison of Line Thinning Algorithms from Digital Geometry Viewpoint," *Proceedings of the 4th International Joint Conference on Pattern Recognition*, pp. 715-719, Kyoto, Japan.

[TD70] Toussaint, G.T. and Donaldson, R.W. (1970), "Algorithms for Recognizing Contour-Traced Handprinted Characters," *IEEE Transactions on Computers*, vol. 19, pp. 541-546.

[Tes86] Tesauro, G. (1986), "Simple Neural Models of Classical Conditioning," *Biological Cybernetics*, vol. 55, pp. 187-200.

[TF81] Tsao, Y.F. and Fu, K.-S. (1981), "Parallel Thinning Operations for Digital Binary Images," *Proceedings of the International Conference on Pattern Recognition and Image Processing*, pp. 150-155, IEEE, Dallas, Texas, U.S.A.

[TG72] Tou, J.T. and Gonzalez, R.C. (1972), "Recognition of Handwritten Characters by Topological Feature Extraction and Multilevel Categorization," *IEEE Transactions on Computers*, vol. 19, pp. 776-785.

[The89] Therrien, C.W. (1989), *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics*, John Wiley and Sons, New York, U.S.A.

[Tho86] Thomason, M.G. (1986), "Chapter 5 — Syntactic Pattern Recognition: Stochastic Languages," in *Handbook of Pattern Recognition and Image Processing*, ed. T.Y. Young and K.-S. Fu. pp. 119-142, Academic Press.

[TIA90] Takahashi, H., Itoh, N., Amano, T. and Yamashita, A. (1990), "A Spelling Correction Method and Its Application to an OCR System," *Pattern Recognition*, vol. 23, pp. 363-377.

[Tol90] Tollenaere, T. (1990), "SuperSAB: Fast Adaptive Back Propagation with Good Scaling Properties," *Neural Networks*, vol. 3, no. 5, pp. 561-573.

[Tou78] Toussaint, G.T. (1978), "The Use of Context in Pattern Recognition," *Pattern Recognition*, vol. 10, pp. 189-204.

[TP86] Torre, V. and Poggio, T.A. (1986), " On Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 147-163.

[Tro76] Troxel, D.E. (1976), "Feature Selection for Low Error Rate OCR," *Pattern Recognition*, vol. 8, pp. 73-76.

[TT89] Tsay, Y.-T. and Tsai, W.-H. (1989), "Model-Guided Attributed String Matching by Split-and-Merge for Shape Recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 3, pp. 159-179.

[TY85] Tsai, W.-H. and Yu, S.-S. (1985), "Attributed String Matching with Merging for Shape Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 453-462.

[Ull69] Ullmann, J.R. (1969), "Experiments with the *N*-Tuple Method of Pattern Recognition," *IEEE Transactions on Computers*, vol. 18, pp. 1135-1137.

[VB64] Vossler, C.M. and Branston, N.M. (1964), "The Use of Context for Correcting Garbled English Text," *Proceedings of the ACM 9th National Conference*, pp. D2 4-1-D2 4-13.

[Vit67] Viterbi, A.J. (1967), "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information*

Theory, vol. 13, pp. 260-269.

[Wal23] Walsh, J.L. (1923), "A Closed Set of Normal Orthogonal Functions," *American Journal of Mathematics*, vol. 45, pp. 5-24.

[Wan80] Wang, P.S. (1980), "Some New Results on Isotonic Array Grammars," *Information Processing Letters*, vol 10, no. 3, pp. 129-131.

[Was69] Wasan, M.T. (1969), *Stochastic Approximation*, Cambridge University Press, New York, U.S.A.

[Wat65] Watanabe, S. (1965), "Karhunen-Loeve Expansion and Factor Analysis," *Transactions of the 4th Prague Conference on Information Theory*.

[WDR81] Wu, A.Y., Dubitzki, T. and Rosenfeld, A. (1981), "Parallel Computation of Contour Properties," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, no. 3, pp. 331-337.

[Wee68] Wee, W. (1968), "Generalized Inverse Approach to Adaptive Multiclass Pattern Classification," *IEEE Transactions on Computers*, vol. 17, pp. 1157-1164.

[Wee70] Wee, W. (1970), "On Feature Selection in a Class of Distribution-Free Pattern Classifiers," *IEEE Transactions on Information Theory*, vol. 16, pp. 47-55.

[Wer74] Werbos, P.J. (1974), *Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences*, Unpublished Ph.D. thesis, Harvard University, Cambridge, Mass., U.S.A.

[Wes86] Weszka, J.S. (1986), "A Survey of Threshold Selection Techniques," *Computer Graphics and Image Processing*, vol. 7, pp. 41-47.

[WG77] Wendling, S. and Gagneaux, G. (1977), "Metric Invariants for Unitary Transformations and Their Application in Character Recognition," *Pattern Recognition*, vol. 9, pp. 233-240.

[WGM73] Widrow, B., Gupta, N. and Maitra, S. (1973), "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 5, pp. 455-465.

[WH60] Widrow, B. and Hoff, M.E. (1960), "Adaptive Switching Circuits," *1960 IRE WESCON Convention Record, Part 4*, pp. 96-104, IRE, New York, U.S.A.

[WHF86] Wang, P.S.P., Hui, L.-W. and Fleming, T. (1986), "Further Improved Fast Parallel Thinning Algorithm for Digital Patterns," in *Computer Vision, Image Processing and Communications — Systems and Applications*, ed. P.S.P. Wang, pp. 37-40, World Scientific, Singapore.

[Wil87] Williams, R. (1987), "Reinforcement Learning Connectionist Systems," Northeastern University, College of Computer Science, Technical Report NU-CCS-87-3.

[Win70] Winston, P.H. (1970), *Learning Structural Descriptions from Examples*, Ph.D. thesis, TR-76, Department of Electrical Engineering, MIT.

[WM91] Wu, S. and Manber, U. (1991), "Fast Text Searching with Errors," University of Arizona, Department of Computer Science, Technical Report TR 91-11.

[WNR74] Weszka, J.S., Nagel, R.N. and Rosenfeld, A. (1974), "A Threshold Selection Technique," *IEEE Transactions on Computers*, pp. 1322-1326.

[Woo70] Woods, W.A. (1970), "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, vol 13, no. 10, pp. 591-606.

[WR83] White, J.M. and Rohrer, G.D. (1983), "Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction," *IBM Journal of Research Developments*, vol. 27, no. 4, pp. 400-410.

[Wri89] Wright, W.A. (1989), "Probabilistic Learning on a Neural Network," *Proceedings of the 1st IEE International Conference on Artificial Neural Networks*, pp. 153-158.

[WS76] Wendling, S. and Stamon, G. (1976), "Hadamard and Haar Transforms and Their Power Spectra in Character Recognition," *Proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, pp. 103-112, Hyannis, Mass., U.S.A.

[WSS89] Waibel, A., Sawai, H. and Shikano, K. (1989), "Modularity and Scaling in Large Phonemic Neural Networks," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 12, pp. 1888-1898.

[WW88] Widrow, B. and Winter, R. (1988), "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *Computer*, pp. 25-39.

[WWC82] Wahl, F.M., Wong, K.Y. and Casey, R.G. (1982), "Block Segmentation and Text Extraction in Mixed Text/Image Documents," *Computer Graphics and Image Processing*, vol. 20, pp. 375-390.

[WZ89] Wechsler, H. and Zimmerman, G.L. (1989), "Distributed Associative Memory (DAM) for Bin-Picking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 814-822.

[Xia86] Xia, Y. (1986), "A New Thinning Algorithm for Binary Images," *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 995-997, IEEE, Paris, France.

[XW87] Xu, W. and Wang, C. (1987), "CGT: A Fast Thinning Algorithm Implemented on a Sequential Computer," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 17, no. 5, pp. 847-851.

[Yam84] Yamada, H. (1984), "Complete Euclidean Distance Transformation by Parallel Operation," *Proceedings of the 7th International Conference on Pattern Recognition*, pp. 69-71, IEEE, Montreal, Canada.

[YG90a] Yair, E. and Gersho, A. (1990), "The Boltzmann Perceptron Network: A Soft Classifier," *Neural Networks*, vol. 3, no. 2, pp. 203-221.

[YG90b] Yair, E. and Gersho, A. (1990), "Maximum A Posteriori Decision and Evaluation of Class Probabilities by Boltzmann Perceptron Classifiers," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1620-1628.

[YM78] Yamada, H. and Mori, S. (1978), "Line-Wise Parallel Operations and Their Application to Handprint Recognition," *Proceedings of the 4th International Joint Conference on Pattern Recognition*, pp. 789-793, Kyoto, Japan.

[YM80] Yamamoto, K. and Mori, S. (1980), "Recognition of Handprinted Characters by Outermost Point Method," *Pattern Recognition*, vol. 12, pp. 229-236.

[YM92] Young, S.R. and Matessa, M. (1992), "MINDS-II Feedback Architecture: Detection and Correction of Speech Misrecognitions," Carnegie-Mellon University, Department of Computer Science, Technical Report CMU-CS-92-119.

[YR82] Yamamoto, K. and Rosenfeld, A. (1982), "Recognition of Handprinted Kanji Characters by a Relaxation Method," *Proceedings of the 6th International Conference on Pattern Recognition*, pp. 395-398.

[YS68] Yau, S.S. and Schumpert, J.M. (1968), "Design of Pattern Classifiers with the Updating Property Using Stochastic Approximation Techniques," *IEEE Transactions on Computers*, vol. 17, pp. 861-872.

[YTF73] Yokoi, S., Toriwaki, J.-I. and Fukumura, T. (1973), "Topological Properties in Digitized Binary Pictures," *Systems, Computers, Controls*, vol. 4, no. 6, pp. 32-39.

[Zad65] Zadeh, L.A. (1965), "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353.

[ZR72] Zahn, C.T. and Roskies, R.Z. (1972), "Fourier Descriptors for Plane Closed Curves," *IEEE Transactions on Computers*, vol. 21, no. 3, pp. 269-281.

[ZS84] Zhang, T.Y. and Suen, C.Y. (1984), "A Fast Parallel Algorithm for Thinning Digital Patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236-239.