Recognition of Cursive Words
Using
String Matching of Contour Chain Codes

A project report
submitted in partial fulfillment of the
requirements for the award of degree of

Master of Technology

in

Artificial Intelligence

by

K. Sri Rama Raju

Department of Computer/Information Sciences
School of Mathematics and Computer/Information Sciences
University of Hyderabad
Hyderabad -500 046
January 1998

CERTIFICATE

This is to certify that the project work entitled **"Recognition of Cursive Words using String Matching of Contour Chain Codes"** being submitted by **K. Sri Rama Raju**, in partial fulfilment for the award of Master of Technology in Artificial Intelligence by the University of Hyderabad, is a bonfafide work carried out at the University of Hyderabad under my supervision. The matter embodied in this project has not been submitted to any other University for award of any degree or diploma.

Mr. Atul Negi, (Supervisor)
Lecturer, Department of CIS,
University of Hyderabad,
Hyderabad - 500 046.

Prof. A. K. Pujari,
Head, Department of CIS,
University of Hyderabad,
Hyderabad - 500 046.

Prof. C. Musili,
Dean, School of MCIS,
University of Hyderabad,
Hyderabad - 500 046.

To my beloved parents, Grandfather

and Grandmother

Acknowledgments

It is a great pleasure to acknowledge my profound sense of gratitude to my guide **Mr. Atul Negi** for his invaluable and inspiring guidance, comments, suggestions, and encouragement throughout the course of this project.

I extend my gratitude to **Prof. A. K. Pujari**, Head, Department of CIS and **Prof. C. Musili**, Dean, School of MCIS for providing me the necessary computing facilities. My heartfelt thanks to all the AI lab staff for their timely cooperation. Unqualified thanks are also due to all my classmates for providing handwritten cursive scripts. Special thanks go to **Mr. Murali** who directly or indirectly helped me in the survey work.

Finally, I am greatly indebted to my family members for their love, affection and moral encouragement throughout my career.

K. Sri Rama Raju.

Abstract

Recognition of Handwritten Cursive Scripts is complex and computationally expensive task. Recognition involves the matching of a reference with a test input. Traditional techniques require complex recognition stages. Several adaptations are required for the feature extraction method to be made suitable for the classifiers. In this report we present an integrated approach where chain codes are used directly for the feature extraction and recognition. The present approach is a holistic classification technique making use of a lexicon of stored reference images. The recognition errors due to segmentation errors are avoided in a holistic recognition technique.

This approach involves Pre-Processing and Recognition phases. Pre-Processing phase comprises of binary image extraction, fixed scale normalization, noise removal, contour extraction and representation as a string of chain codes. A novel contour splitting technique is proposed to solve the registration problems in recognizing shapes for the cursive word recognition thereby reducing the number of computations at the recognition stage. Recognition phase comprises of finding a set of matches for the unknown string's upper contour with those of the references' upper contours stored in the dictionary This produces a candidate set for which the same matching procedure is applied but using the lower contours of the reference set as well as that of the unknown string Failing to find a unique match, Cyclic invariant Edit distance computation algorithm is applied to evolve a match. A dictionary of 696 samples comprising of 29 different numerals each one having 24 variants is employed, and is found to provide 95% correct results.

Contents

Chapter I

Introduction

1.1 Introduction

Since the advent of a digital computer, there has been a constant effort to expand the domain of computer applications. Some of this motivation comes from the sheer challenge of building or programming a machine to perform novel and more demanding tasks. Some of the motivation also comes from finding more efficient ways of doing cognitive tasks. Both of these are found in pattern recognition and image processing aspects of machine perception.

The ability for machines to perceive the environment has been very limited. We now move beyond having computers read punch cards or magnetic tapes to having them read hand printed characters or analyze Bio-Medical photographs. The domain of the problems move from sensing data to much more difficult problems of interpreting the data. Psychological and physiological studies have given interesting facts about animal perception, but the present understanding is insufficient for us to emulate their performance with a computer. Hand written recognition is one of such challenging perceptual tasks. Many of the challenging tasks involve pattern classification and cognition.[Duda & Hart:73]

For more than thirty years, researchers have been working on handwriting recognition Recognition of hand written scripts has numerous applications in day to day life, such as Document Analysis, Reading of bank checks, Recognition of address for postal sorting, Reading of credit card slips, Writer identification, Signature verification, etc.

The most common design issues in developing effective systems for handling handwritten scripts are :

- Implementation of efficient image processing routines
- Development of efficient recognition algorithms.

Conventional image processing techniques deal with two dimensional images collected and digitized using scanners. Significant processing time is spent in image handling due to the nature of two dimensional representation of objects making it inappropriate for implementation in realistic applications involving human interaction The image representation methods have been developed to minimize amount of data without loss of information, thus increasing speed.

Over the last few years, the number of academic laboratories and companies involved in research on handwriting recognition has continually increased. Simultaneously, commercial products have become available. This new stage in the evolution of handwriting processing results from a combination of several elements.

- Improvements in recognition rates
- Use of complex systems integrating several kinds of information
- Choice of relevant application domains

- New technologies such as high quality high speed scanners and inexpensive powerful CPUs [Faure & Lecolinet:96].

In this project keeping in view the efficiency of image handling, chain code based image representation is employed. Image handling procedures commonly used in most handwritten recognition systems including preprocessing and feature extraction are here implemented using the same efficient chain code based techniques. Preprocessing involves scanning, storing, noise removal and normalizing. The features are extracted by proper thresholding of the images. The main requirement is a binary image file obtained by proper thresholding of a gray image obtained from a scanner. Later the recognition is done by string matching techniques. In the following sections the OCR system is described in detail.

1.2 Optical Character Recognition System (OCR)

The written form of language is contained in printed documents, such as newspapers, magazines and books, and in handwritten matter, such as found in notebooks and personal letters. Given the importance of written language in human transactions, its automatic recognition has practical significance [Srihari & Rohini 96]. There are various types of paper documents such as documents printed from printers, typewriters, hand written documents, photocopied documents or a combination of all the above. In the case of documents printed from typewriters, the text will usually be limited to a single font and size. Documents from printers usually have limited variations of type size and fonts. In the case of hand written documents, the text, usually is cursive, and is not of a fixed size or font, and varies from writer to writer. Another major constraint is the category of handwriting that recognition systems are able to handle [Lecolinet & Baret:93]. Various categories are:

- Boxed discrete characters
- Spaced discrete characters
- Run - on discrete characters
- Pure cursive script writing
- Mixed cursive and discrete

The last two categories of the above mentioned styles are more difficult to recognize because the letters run together. Apart from the variation from writer to writer, there can be variation of slope, size, and style from sentence to sentence. It may even be the case that the same writer, depending on the mental state, may vary the style of writing from time to time. Inspite of all these variations, some invariant features were recognized from human analysis of hand written text.

Computer analysis and understanding of paper document is a challenging task. It requires the development and integration of techniques in several areas of Computer Vision, Artificial Intelligence, Knowledge Representation and Natural Language Processing

Computer document analysis requires the acquisition of document images, segmentation of the document images into individual text rows and segmentation of each row into individual words Now the problem can be approached in two ways. One is at word level and the other is at character level. If the document is printed, then the best method is to do character or letter analysis, and if hand written, then it can be either character level or word level, using dictionary lookup. After recognizing the individual letters, the letters are integrated to generate words, and words into sentences, and the whole information is retrieved. The system that performs the above mentioned task is an Optical Character Recognizer (OCR).

Character Recognition or Optical Character Recognition (OCR) is the process of converting scanned images of machine printed or handwritten text (numerals, letters, and symbols), into a computer-processable format (such as ASCII). There can be different types of OCR systems depending on the nature of application they are built for. For printed document recognition, a simple OCR system that can recognize isolated characters is sufficient, but for applications like check reading, OCR systems that can recognize cursive script, as well as printed text are needed. Further, depending upon the complexity, the OCR systems can be classified as :

- Fixed font CR systems
- Multi font CR systems
- Omni font CR systems
- Cursive Script Recognizers

The last type is the most complex and difficult to design They are expected to recognize cursive handwriting as well as printed text.

Optical Character Recognition is generally divided into two categories: on-line recognition and off-line recognition. On-line recognition is performed concurrently during the writing process. This is usually achieved through pen-based interfaces, where the user writes with a special pen on an electronic tablet. The advantage of such methods is that they capture the temporal or time based information of writing. This includes information about the strokes number, order and direction. Another advantage is that the trace of the pen is one pixel wide, removing the need for skeletonizing or thinning techniques.

Off-line recognition is performed after the writing is completed, and cannot take advantage of temporal information. The already written text is usually captured by an optical scanner for recognition. The basic input unit of off-line recognition is a bitmap, meaning that off-line algorithms operate on static data. Some claim that on-line recognition is superior to off-line recognition, and have run both algorithms on the same data to show this. Off-line recognition has the advantage of being applied on any previously written document, without the need of special writing equipment. Historically a very large base of written material exists such as land records, sale deeds etc., which involve problems of off-line handwriting recognition. Hence the need for implementing effective off-line handwriting recognition systems. In this project henceforth the subject of study is off-line recognition

1.3 Working of OCR

The working of a general optical character recognition system comprises of three consecutive stages[Srihari & Lam:95] namely

- Pre-processing phase
- Recognition phase
- Post-processing phase

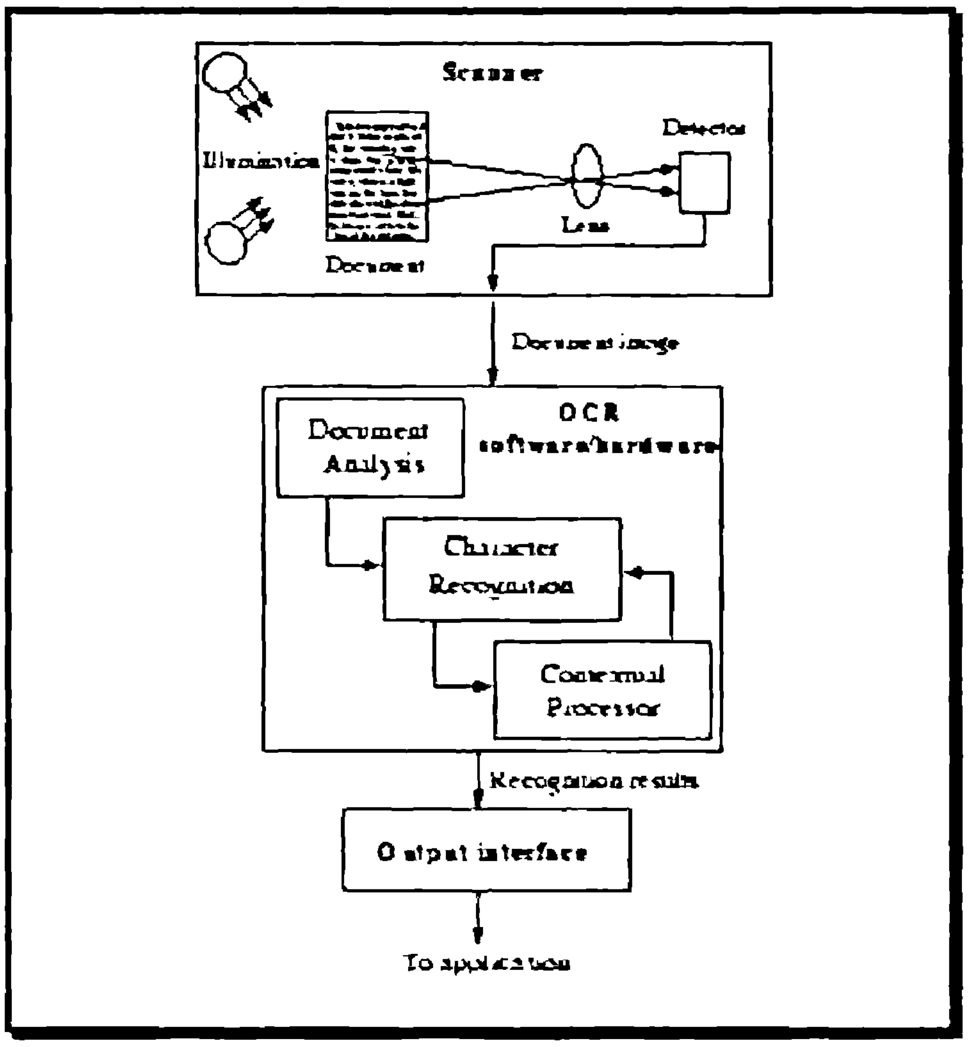


Figure 1.1 OCR involving Document analysis, Recognition and Contextual processor

1.3.1 Pre-Processing

The preprocessing phase involves collection of various samples of images, scanning, storing and retrieval of images, thresholding, normalizing or scaling and finally noise filtering.

The samples are from data written by several people. The sample images are then scanned by an optical scanner and are stored onto magnetic media. Binary images are sufficiently informative for most OCR applications. Binary images are also easier to handle in terms of image manipulation procedures. Since the scanner outputs a gray image thresholding is needed to convert the gray image into binary image. For some applications the images may need smoothening to remove jaggedness caused in the scanning process. It may be necessary to remove skew in words (i.e., skew correction), but most of the methods work inspite of skew being present. The images are of varying sizes, so normalization is done to adjust the image size to a fixed size. Processes that attempt to compensate for poor quality originals and/or poor quality scanning include image enhancement and noise removal. Image enhancement methods emphasize character versus non-character discrimination. Noise removal erases portions of the image that are not part of the characters [Srihari & Lam:95]

1.3.2 Recognition phase

It seems reasonable to look for features that are invariant. Normalizing transformation is used to regularize the variation to a common form suitable for image handling and recognition.

The essential components in a character recognition algorithm are the feature extractor and the classifier. Feature analysis determines the descriptors, or feature set, used to represent all characters. Classification is performed by comparing an input character image with a set of templates (or prototypes) from each character class. The similarity score between the input character and the template is computed. The similarity is high when a pixel in the observed character is identical to the same pixel in the template image. If the pixels differ, the similarity may be less. After all templates have been compared with the observed character image, the input character is assigned the label of the template with the highest similarity score. Recognition approaches can be divided into two general types, i.e. analytic and holistic.

The analytical approaches attempt to identify letters, or parts of letters that constitute the whole word. Since letters in a word are connected by ligatures, it is generally impossible to identify the letters uniquely without reference to context. The usual method, is to partition each word into sub-letter segments, or word units, a number of which can constitute a single letter. Analytical methods differ in the way word unit segmentation is performed, giving rise to two main categories:

- Explicit Segmentation
- Implicit Segmentation

Explicit segmentation includes external segmentation of the word into smaller units, and individual recognition of these units. Implicit segmentation methods, due to their inherent reliance on recognition, are computationally very expensive. Contextual post processing and dictionary look up in this case too may be ruled out.

The holistic paradigm in handwritten word recognition (HWR) refers to the paradigm of treating the handwritten word as a single, indivisible entity and attempting to recognize it using features of the word as whole. The main advantage of holistic methods is that they avoid word segmentation. Their main drawback is that they are related to a fixed lexicon of word descriptions [Madvanath:96]. As these methods do not rely on letters, words are directly described by means of features, and adding new words to the lexicon require human training or the automatic generation of word descriptions from ASCII words. These methods are generally based on dynamic programming (edit distance, DP-matching, model-discriminant Hidden Markov Models etc.). One way to recognize the word is to find an external contour of the whole word, find it's bounding box and use a matching technique to find the similarity, there by recognizing the word. The matching technique that is generally used is edit distance technique [Bunke & Bühler:93]. The difference lies only at the contour level. One representation of contour uses a stick length and use fixed length lines to construct the contour while another one considers pixels and constructs the contour using 8 directional chain codes.

An edit distance operation can be used as a recognition technique to select the template word having the most similar features to gauge the similarity of a template to a input sample. This method is useful if the size of lexicon is small.

1.3.3 Post-Processing Phase

The post processing techniques are applied after the recognition stage. Output from the recognition stage needs to be collected, disambiguated, ranked and verified. Contextual and probabilistic information, including orthography of the language and dictionary lookup are used. Most of the techniques can be classified into one of the two categories.

- Statistical Representation of Contextual Knowledge
- Structural Representation of Contextual Knowledge

The two approaches have been referred to as bottom up (Data driven) and top-down (concept driven) methods respectively. Statistical representation of contextual knowledge contains models of word generation process that is closer to the Artificial Intelligence framework, and is based on deterministic representation of contextual knowledge. Examples of top-down approaches are, use of dictionaries, language syntax and semantics [Riseman & Hanson:74],[Sinha & Prasada 88],[Belaid 96].

In the case of implicit segmentation technique, word level knowledge can be brought in during the recognition-segmentation stage. The contextual knowledge can be used in a statistical way by using a lexicon. Statistical representation has become very popular these last few years with the use of Hidden Markov Models (HMM). In these models, contextual information is represented by means of transition probabilities.

1.4 Why Edit Distance?

Our main aim is to recognize the word as a whole instead of recognizing each character. The recognition problem is reduced to computation of edit distance. So the edit distance seems to be a good method because slightly different strings (due to variations in writing style) would be correctly recognized. A small part of the word, not correctly written, does not make drastic changes because, it will have a cost of a few edit operations only. This is the main advantage of edit distance over other methods. The only disadvantage is that its complexity increases with the size of the dictionary. The time taken for arriving at the result increases and at the same time incorrect results may be obtained

In this project, recognition of a limited set of words is considered, that is the English numerals form one to twenty, and multiples of ten upto hundred and thousand.

1.5 Usage of Lexicon

Concept driven algorithms proceed with an expectation of the likely input string, and fit the data to meet this expectation. Thus, a dictionary provides the expectation of syntax and semantics necessary for a concept. Such a recognizer can be useful for recognition of the legal amount in bank checks.

Global recognition approaches, which try to recognize the word as such, without trying to recognize individual characters need a dictionary to compare the input and the templates, to propose the most probable word. An incremental search for a given string can help prune the search for hypothesized words by predicting the word according to the current state of search. Given a sample in which it is unknown whether or not an error is present, the most straight forward scheme is to lookup the word in the dictionary. If the word is in the dictionary it does not necessarily mean that no error has occurred. An error may have transformed one word into another. The organization of the dictionary also matters a lot. In most of the applications, it would be sequential. It may also be organized as a trie. Organizing the dictionary as a trie is the most efficient way. In our problem, since the distance of the unknown string with all the known strings (templates) is computed, the dictionary is organized sequentially

1.6 Motivation

Researchers contend that human eye processes the whole sequence of letters simultaneously, rather than individual characters sequentially, for recognition purpose. In this light, character based identification as done with most of the classical pattern recognition schemes is not the optimal approach. Human beings perform character recognition better than OCR systems currently available by at-least an order of magnitude in error rate. This is due to the human knowledge of contextual factors such as letter sequence, word dependency, sentence structure & phraseology, style, subject matter. Inferences, associations, guessing, prediction and imagination, also aid the natural process of reading.

It is clear that if computer programs are to reach the expert level of human ability in text recognition, they need to be able to effectively integrate diverse contextual knowledge sources about the text, as well as the knowledge about the textual errors that are likely to occur.

Equally important is the fact that edit distance and HMM based approaches can give rise to a number of hypothesized words for the word to be recognized. Syntax and semantics can be used to eliminate the wrong hypotheses later. This is very similar to human reading process. This is possible by suggesting those words that have an edit distance within a threshold from the input string In our approach previous knowledge is simulated by the dictionary.

1.7 Example

Before going into the details of organization of the report, let us have a feel of what is an edit distance, by taking up a small example. Here only a feel of what edit distance is, is presented. An instance of many possibilities of computing edit distances is given.

One thing to be remembered is that, the contours are represented using Freeman's 8-directional chain codes, and that a set of chain codes as a whole is considered as a string.

Let X and Y be two strings that are to be matched using edit distances

X : 014302 of length(X)=6 ; Y : 0303017 of length(Y)=7

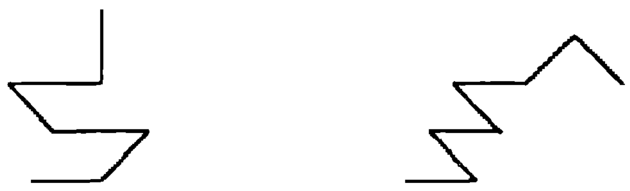
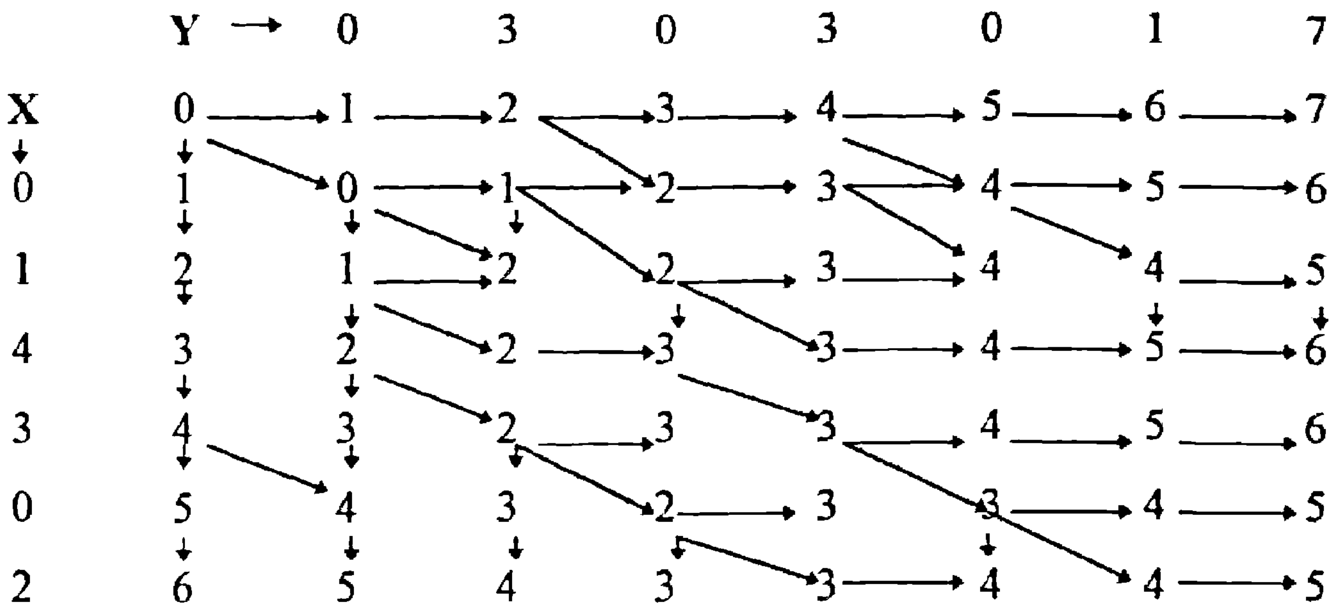


Figure 1.2 : Contours of X and Y

It is assumed that :

- Insertion cost is fixed to 1, i.e., replacing empty word with any chain code in Y,
- Deletion cost is fixed to 1, i e., replacing any chain code in X with empty word,
- Substitution cost is not fixed, but varies, which is further discussed in detail in Chapter Four.

Then distance between the two strings X and Y is obtained by calculating the matrix D as follows :



	Y →	0	3	0	3	0	1	7
X	0	1	2	3	4	5	6	7
0	1	0	1	2	3	4	5	6
1	2	1	2	2	3	4	4	5
4	3	2	2	3	3	4	5	6
3	4	3	2	3	3	4	5	6
0	5	4	3	2	3	3	4	5
2	6	5	4	3	3	4	4	5

Table 1.1 Computation of Matrix D

Distance is D[6,7] = 5.

All possible edit operations to arrive at the final cost are :

sesisssi cost is : 0+1+1+1+0+0+1+1 = 5 (total 8 edit operations)

sisesssi cost is : 0+1+1+1+0+0+1-1 = 5 (total 8 edit operations)

So the similarity measure between X and Y is 5

1.8 Organization of the Report

In Chapter 2 formal definitions and topics regarding the related work done in preprocessing such as, extracting binary image from gray level image, normalization and filtering, extracting contours (chain code representation methods, line segment extraction methods), methods for splitting contours, etc., are discussed.

In Chapter 3 a few methods for string matching using edit distances are discussed. This Chapter deals with formal methods for computing edit distance, with a special attention towards Levenshtien distance and Bunke's method of computing edit distance.

In Chapter 4 our method of string matching using edit distances along with descriptions of algorithms and data structures for all the major functions such as binarization of gray image, filtering, normalizing, extracting contour, splitting of contour into upper and lower contours (least mean square fit line), matching of upper and lower contours, rotational invariance in contour matching, etc., is given.

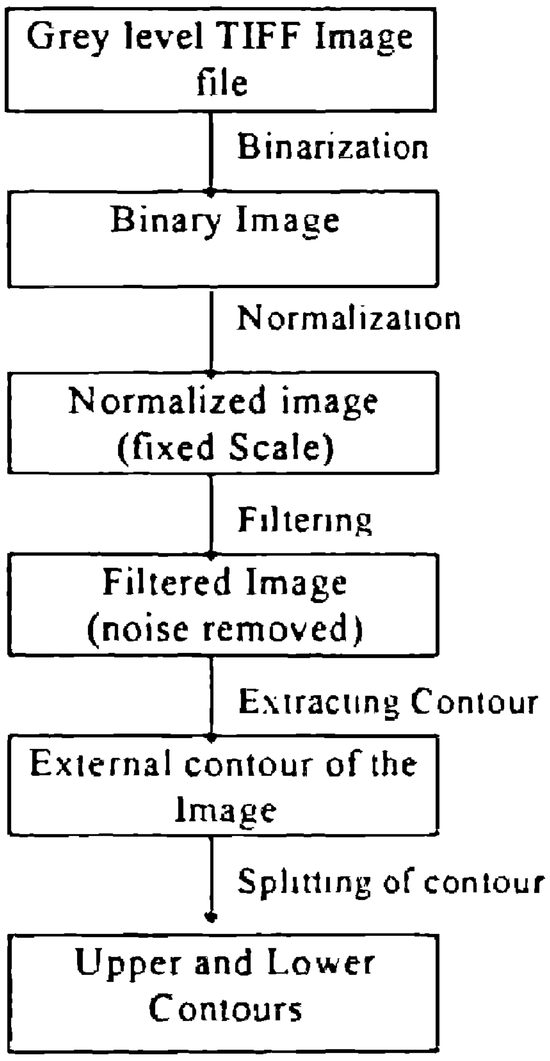
In Chapter 5, the results that are obtained by applying the proposed method are given. In this section only those samples that are poorly written, wrongly recognized and examples that can be misinterpreted are given Also statistics of the minimum and maximum costs for each of the input sample with its own class of samples stored in the dictionary are also given.

In Chapter 6 conclusions as well as future directions for the project is given. In the Appendixes, the information that is present in a gray level TIFF image file, samples that are taken as a Data set and the steps for creating a new Data set are given.

Chapter II

Pre Processing Techniques and Survey

In this Chapter a brief description about the survey work relating to the pre-processing is given. The pre-processing comprises of various sequences of operations as shown in the figure 2.1.



Grey level TIFF Image file
Binarization
Binary Image
Normalization
Normalized image (fixed Scale)
Filtering
Filtered Image (noise removed)
Extracting Contour
External contour of the Image
Splitting of contour
Upper and Lower Contours

Figure 2.1 : Pre-Processing Stages

2.1 Terms and Definitions

Syntactic Pattern Recognition : Also called as structural pattern recognition, the classifiers are designed, trained and tested by representing the patterns symbolically using primitive or elementary data structures like trees, graphs, arrays and strings [Oommen & Kashyap:96].

Statistical Pattern Recognition : In this the patterns are represented using numerical features. The classification is done based on the probabilities such as apriori probabilities, class conditional probabilities and posteriori probabilities[Duda & Hart:73].

Bi-Level Image : A display of a page of text is a bi-level image, also called as binary image or black and white picture. Such images can be represented in a matrix form with one bit per element (pixel) which is either 1 or 0 (black or white).

Gray level image : In such images each picture element takes on a value between zero and the number of gray scales or gray levels. There can be 16 shades of gray or 256 shades of gray in an image[Phillips.94]

Thresholding : This technique is based upon a simple concept A parameter 'θ' called the brightness threshold is chosen and applied to the image a[m,n] as follows:

if **a[m,n]** >= θ then **a[m,n]**=object = 1; else **a[m,n]**= background - 0, This assumes that light objects on a dark background are of interest. For dark objects on a light background it would be:

if a[m.n] < θ then a[m,n] = object = 1 else a[m,n] = background = 0

The output is the label "object" or "background" that, due to its dichotomous nature, can be represented as a Boolean variable "1" or "0". In principle, the test condition could be based upon some other property than simple brightness.

Normalization : Also called as scaling or resizing, is the process of compressing or enhancing an image to a fixed size or scale.

Noise : Noise is the undesired information present in the image, which interferes with the correct interpretation of the image either by human beings or computers.

Filtering : Filtering is the process of removing noise occurring in the image when it is scanned (i.e., image enhancement).

Chain Code : This representation is based upon the work of Freeman In this the vector joining two successive points (pixels) is assigned one symbol from among a finite set. A common chain code that is used generally is 8 directional chain code as in figure 2 2 [Phillips.94].

3	2	1
4		0
5	6	7

Figure 2.2 8-directional chain code

Connected components : A set of pixels **S** is said to be connected if for every pair of pixels **c** and **d** in **S**, there is a path whose first and last elements are **c** and **d** respectively[Pavlidis:82].

d-neighbor : Two pixels are said to be direct neighbor's if the respective cells share a side [Pavlidis:82]. (i.e. horizontal or vertical)

i-neighbor : Two pixels are said to be i-neighbor's if the respective cells touch only at a corner[Pavlidis:82]. (i.e. Diagonal)

Contour : The contour or i-contour of a connected set of pixels **R**, is defined as the set of pixels in **R** which have at-least one d-neighbor not in **R**. The d-contour of **R** is the set of all pixels in **R** which have at least one neighbor not in **R**[Pavlidis:82].

String : It is a sequence of symbols used to represent both the known shapes as well as unknown shapes. The symbols can be ordinary numbers or angular values or characters or chain codes[Bunke & Buhler 93].

Test Set : It is the set of unknown images or strings that are taken as input and are to be recognized using some matching technique with the data set

Data Set : It is the set of known sample images or strings that are used in the matching process to recognize the unknown input

Matching : Matching is the process of comparing two or more structures to discover their likeliness or difference. The structures may represent objects, including words, phrases, and strings etc.

Distance Metric : It is a measure of the degree of association or likeliness between the object's attributes and other characteristics.

$d(x,x) = 0$

$d(x,y) >= 0$

$d(x,y) = d(y,x)$

$d(x,y) <= d(x,z) + d(z,y)$ where **x, y, z** are elements;

also $d_p = [\sum_{i=1}^{n}[x_i - y_i]^p]^{1/p}$.

2.2 Various File formats and Extracting of Bi-level Image from Gray Scale Image

The first step in any image processing or pattern recognition task is to obtain an image. The programmer needs a simple method of obtaining image data in a standard, usable format. Images are usually obtained from digital cameras or scanners. It is possible to set the resolution, attributes (i.e , color or gray level) and format of the image at the time of scanning them

The frequently used image file formats are BMP, GIF, JPEG, PCX, PNG, TGA and TIFF.

BMP (Bitmap image file format) : Bitmap is the simplest form of raster image file formats available and many variations of the bitmap format exist. They typically consist of a header specifying height and width for the image, followed by a long string of binary encoded numbers, each of which represents a color for an individual pixel. Bitmaps can come in a number of color depths, ranging from 2 colors to millions. As the color depth increases, so too does the size of the bitmap, even though the dimensions of the picture do not change. Bitmaps are loaded quickly into video memory and displayed quickly because there is no decoding involved in the display of these files. Bitmaps do not adjust easily to resizing and editing because of their strict pixel by pixel nature. Thus they do not work well in elaborate drawing or painting packages.

The bitmap image file consists of three parts, a file header, bitmap information header, which includes color palette information and the actual image data. The Bitmap image file header consists of information like file type, file size and offset to the image data. Bitmap information header consists of the following: width, height, bytes per pixel, type if compression, image size, number of colors used and a color information table. Run-length encoding is used in compressed bitmap images. This is the windows native format. In-order to use this format efficiently, we need to use the functions of windows, which does not support easy and direct manipulation of image data.

GIF (Graphics Interchange Format) : This format was designed to deliver high resolution images over the internet. To decrease the waiting time for images to come across the internet, CompuServe devised a compression scheme for bitmaps. The GIF compression, unlike JPEG compression is non-lossy It uses a proprietary encoding/decoding scheme called LZW (after its inventors Lempel, Ziv, Welsh). It is relatively complex to encode or decode over the RLE (Run length Encoding) compression scheme, which gives a larger compressed file size than GIF. A gif file consists of a header and the encoded data. When the image is viewed, the software uses the LZW scheme to decode the file into a bitmap for display. Similarly, when a bitmap is saved as a GIF, the bitmap is encoded and compressed. This compression works well and led to the popularity of the GIF format. Enhancements to GIF are, interlaced GIFs, transparent GIFs, multiple images in a single file, interleaving scan lines (as in PCX format), and animated GIFs. GIF supports only 256 colors, which is a great limitation for photographs and high resolution images.

JPEG (Joint Photographic Experts Group) : It was developed by ITU, ISO and IEC. It is a standard for continuous-tone still pictures. For photo-realistic images, the JPEG format delivers high quality image with powerful compression Depending on the amount of compression or quality requested, it involves certain amount of rounding off, and it is here, that some of the loss takes place A JPEG image can have a compression, as high as 20:1 A GIF specifies the color of each pixel in an image, while JPEG is an interpretive format which describes the appearance of an image in deliberately imprecise terms Conversion to JPEG therefore always entails "loss" in that, the image data is not recorded verbatim The format optimizes quality by concentrating degradation where it does the least damage. Since human eyes perceive variations in brightness more than variations in hue, the former are recorded with care, and the latter with some liberty, all to a degree of precision set by the user. Unlike GIF, all JPEG images have virtual 24-

bit color depth no matter what the original color depth is. An advantage to the JPEG method besides compression is that it is adaptable to every platform. The standard ("baseline") JPEG must be loaded from top to bottom, since it records all the samples for one horizontal segment at a time. A progressive JPEG records samples for the entire image starting with the largest blocks. Like an interlaced GIF, this lets the entire picture be displayed and resolved when loading. One JPEG mode that is generally used is a lossy sequential mode.

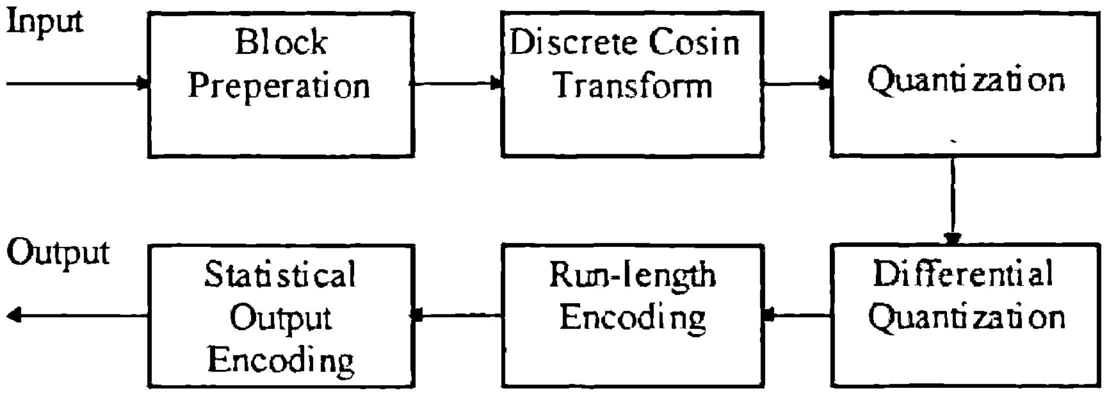


Figure 2.3 : Lossy sequential JPEG mode

JPEG is recommended for compressing large pictures that are originally scanned as TIFFs or BMPs. Line drawings are not to be compressed using JPEG Instead, GIF or PNG are to be used.

PCX : The PCX image file consists of a PCX header (128 Bytes), the image data and a color palette information (for images with 256 colors and above) The PCX header comprises of information like bits per pixel, x and y dimensions, x and y resolutions, line buffer size and the color palette information (for images with 16 colors or below) Runlength encoding is used for compression Unlike in BMP format, PCX stores the R, G and B values of pixel colors in separate rows For example, to decode a 16 color PCX image, four scan lines are read, and one bit from each scan line is combined, to obtain the corresponding pixel's index value in the 16 color palette.

PNG (Portable Network Graphics): The PNG file format is a new (and better) lossless graphics format introduced to replace the GIF file format. The main reason for the implementation of PNG was the age of the GIF format and the controversy over the LZW compression algorithm that GIF uses. This format supports a color depth of upto 48 bits per pixel.

TGA (Targa file format) : It is developed by Truevision Inc.. This format supports 16, 24 or 32 bits per pixel. This format is mostly used with frame grabber boards.

TIFF (Tagged Image File Fonnat) : It is developed by Aldus.

The standard image file fonnat used in this work is the TIFF fonnat. The TIFF image file format (Gray scale) is discussed in more detail in Appendix A. Once all the relevant infonnation is read from a gray level image file it can be easily converted into bi-level image by using the algorithm which is discussed in detail in Chapter Four.

2.3 Normalization and Filtering

Once the binary image is obtained, processing of this image can be carried out to further extent. But our present work is fixed to pixel level So if the dimension of the images is not constant and is varying from image to image, the number of computations that are to be done later while matching would be very high. Also the distance measurement would be large, even though they may have the same shape. There are many disadvantages by having varying dimensions one of which is mentioned above and the others are delay in matching with a large data base if the images are large, Consumption of space, processing delay, etc. So normalization of the image is needed. The theory behind normalizing is simple. Image can be blown up or condensed. There are two methods in normalizing. One is keeping the aspect ratio constant and the other is by varying the aspect ratio. Since the collected images are of various sizes and shapes. normalization without keeping aspect ratio is considered in this work.

Once the normalized binary image is obtained it can be processed. Due to inevitable disturbances in the scanning process noise is incorporated into the image. So in-order to get the exact image, noise need to be removed. The noise that is encountered in this work is due to texture of paper, quality of writing devices and small portions of other images in the current image There are many techniques available that are used to remove noise, one of which is filtering technique. The filtering technique that is employed in this work is discussed in Chapter Four.

2.4 Extraction of contour from Bi-level Image

In the hand written recognition problem only the black portions of the bi-level image (excluding all the noise) are of interest There are various methods and techniques that are of interest and are used in extracting only the black portions of the image, a few of which are zone finding operations that determine three main zones of script viz. the middle zone (where the body of all the letters resides) and upper and lower zones corresponding to ascenders and descenders, and contour extraction methods[Bozinovic &Srihari:89]. Recent work, contour extraction methods are of concern. Again in the contour extraction process the representation plays an important role. There are two types of representations namely chain code representations and line segment representation. In the next few sections contour extraction based on the above two representations is dealt.

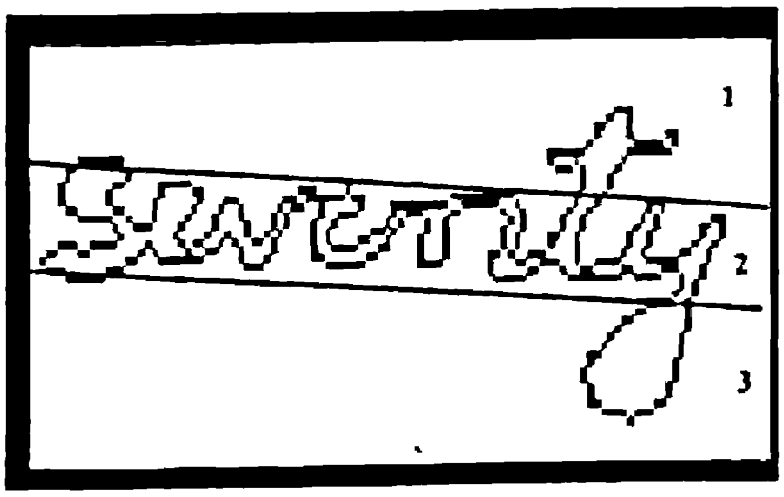


Figure 2.4 : Contour with all the three zones 1. Upper Zone 2. Middle zone 3. Lower zone (Resolution 320 x 200)

2.4.1 Chain code representation

It is the easiest of the two contour representation schemes. A bi-level image can be thought of as a two dimensional array with 1's and 0's that can be easily represented in x-y coordinates. The process of then finding the chain code from x-y coordinates is trivial and is straight forward. There are 3 types of chain codes namely 4-directional chain code, 8-directional chain code and 16-directional chain code. For our problem, 8-directional chain codes are employed.

There are various methods for extracting contour from a bi_level image using chain codes, two of which are discussed in the next sections.

```
70070000010010117700000010117767001121121010765666607000001010211
10076566600012222101000007756654667000010100770000000107000100765
54445454555546567660700071001012122122122212121211224546554544443
12121244655565544454322212111121110000001245444432224455465544444
67001766565565465445431212233444444444454444444446554545444321100 12
43544434454556554323234556554533111124344446566544334321013244444
4546566707000077544454344545
```

Figure 2.5 : 8-direction chain code for figure 2.4

2.4.1.1 Dynamic Approach for finding the contour of Bi-level images

This model [Chow et. al:94] employs the order of the checking sequence depending on the occurring probabilities of neighboring pixels. The neighboring pixel that lies in the same line linking the preceding and the current boundary pixels has the highest probability, about 50%. The two pixels that lie immediately clockwise and counter-clockwise from the pixel with the highest probabilities, about 12.5% each. The chain code of the current boundary pixel is expressed as the direction number from the current pixel to the next. The values of direction number are arranged from 0 to 7, starting from the nearest right and proceeding in counter-clockwise manner, as shown in Figure 2.6 by circled numbers.

It uses the probability-based list (P-list); [0,1,-1,2,-2,3,-3] to determine the search sequence. In the P-list, the absolute value of each number represents the priority in the search sequence. The lowest absolute value has the highest priority and the positive number is searched before the negative if their absolute values are the same.

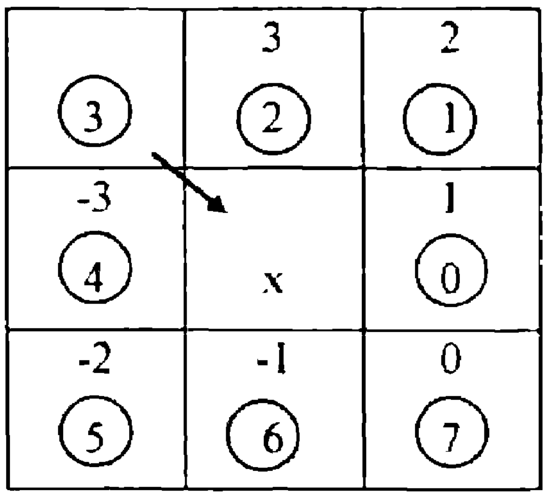


Figure 2.6 : Chain codes with corresponding weights when previous direction is 3

The neighboring pixel that is located in the same direction linking the processing and the current pixels has the highest probability The probabilities of the remaining seven neighboring pixels are obtained from the distances to the highest probability pixel In other words, a pixel nearer to the highest probability pixel has a greater probability If the probabilities of two pixels are the same, the pixel in the counter-clockwise direction is searched first Figure 2 6 shows an example, in which the current pixel is denoted by x and the direction number of the preceding pixel is 3. Thus the next most probable boundary pixel is the cell of direction number 7. The values of the P-list corresponding to directions [7,0,6,1,5,2,4] are [0,1,-1,2,-2,3,-3].

After the next boundary pixel is found, the current pixel then becomes the preceding one, and the next pixel becomes the current one. The search sequence for the new current pixel is determined by first adding each number in the P-list to the direction number from the preceding pixel to the current pixel. Then, the function mod 8 is applied to the sums, which makes a search list (S-list). The S-list indicates the search sequence for the new current pixel. Contour is represented by its starting point and it's chain codes. A dynamic array is used to store chain codes.

Algorithm :

Step 1 : [Initialization] Set contour no to 1, direction number to 0, P-list to (0, 1, -1, 2, -2, 3, -3) and chain code to 0
Step 2 : [Follow all contours] Repeat steps 3 to 6 until all the attribute values of pixels are 0.
Step 3 : [Finding the starting point of the current contour] Set the pointer to the chain code of the current contour to null. Find the first pixel which has an attribute value of 1(this point is the starting point of the contour) Store the coordinate value of the starting point in the bucket.
Step 4 : [Finding the direction from the current pixel to the next boundary pixel] Let chain code be the direction number from the current pixel to the next boundary pixel
Step 5 : [Follow the current contour to find its chain code] Repeat Step 6 until the next boundary pixel is the starting point of the current contour or there is no next boundary pixel.
Step 6 : [Find the first pixel with an attribute value of 1] Adding chain code to each element in the P-list, moded by 8 Then save the result in a S-list

2.4.1.2 Contour following(Duda and Hart)

In this subsection, the contour following technique presented by Duda and Hurt is discussed in detail[Duda & Hart:73].

The picture may be simplified by representing objects in the picture by their outlines. Contour following involves tracing out the boundary between a figure and its background. The image is an analog image with two levels of gray. Imagine a contour following bug that scans across a picture until it enters a dark region. It then curves to the left until it leaves the figure, at which point it immediately begins to curve to the right. Repeat this process, the bug will follow the figure boundary in the clockwise direction until it returns to the neighborhood of the starting point. The set of transition points between figure and background can be taken to represent the contour

Several observations can be made about the above algorithm. First, The radius of curvature employed by the bug determines the "resolution" of the contour follower. The bug will obviously not detect curves that are sharp compared to this radius of curvature Similarly if there are holes in the figure just inside the boundary the bug can wander into a hole and produce erroneous transition points. Finally, the algorithm as stated is not well defined, since the centers of each curved section of the trace have not been specified

The digitized version of the algorithm can be simply and precisely defined as follows :

Step 1 : Scan the picture until a figure cell is encountered

Step 2 : Then If you are in a figure cell turn left and take a step.
Step 3 : If you are in a back ground cell turn right and take a step.
Step 4: Terminate when you are within one cell of the starting point.

Some of the disadvantages of this algorithm is that, it knows about 4-connectedness and 8-connectedness, and employs 4-connectedness in defining the set of figure cells connected to the cell it enters. Also that the set of transitions points determined by the algorithm depends upon the initial scan. Similarly a hole in the figure that is 8-connected to the background creates an ambiguity in the set of transition points that the algorithm determines. This is due to the excessive simplicity of the algorithm. Specifically only three bits are used at each point to decide which way to turn : two bits tell from which direction the cell was entered and the third bit indicates whether the cell is figure or background

Two major conditions must be fulfilled if the contour following algorithm is to be successfully employed One is proper thresholding of the gray level image. The other is the figure has no spurious gaps in it.

2.4.2 Line Segmentation or Line Description

The general approach will be to fit the figure by one or more straight lines (or perhaps polynomial curves) There are two problems where to break the original figure into segments, and how to fit a line to each segment[Duda & Hart:73]. There are a number of methods for accomplishing this The main idea behind all these techniques is to transform the original figure into a new domain in which collinear subsets of the figure fall into clusters

The other technique is line segmentation. This partitions the figure points or curve into a number of consecutive line segments such that each segment can be reasonably approximated by straight line. Two methods for line segmentation are discussed in the next section.

2.4.2.1 Iterative End-Point fits

This method is simple and goes this way. Given a set of **n** figure points (x_i,y_i) **i = 1,2,....n**, an initial line is fit, call it AB, by merely connecting the end points of the set (i.e. the left and right extremes). The distance from each point to this line is computed; and if all the distance are less than some present threshold, the process is finished. If not, find the point furthest from AB, call it C, and break the initial line into the two new lines AC and CB. This process is then repeated separately on the two new lines, possibly with different thresholds It has two drawbacks both of which can be traced to the fact that it can be strongly influenced by single points. The most serious drawback is that a single wild point can drastically change the final result By preliminary smoothing and noise removal good results can be achieved[Duda & Hart 73]

2.4.2.2 Points of maximum curvature

This is a different kind of segmentation problem Given a smooth contour (that might be obtained, for example from a contour follower) it is broken at points of high curvature and the break points are connected by straight lines Mathematically this approach is related to the so-called intrinsic equation of a curve. The intrinsic equation of a curve is defined to be its curvature as a function of its arc length. Hence the method under discussion is essentially equivalent to representing the intrinsic equation by its extreme points.

2.5 Splitting contour into upper and lower contours

In most of the character recognition problems the contour as a whole is taken into consideration while doing string matching. Also different criteria like robustness with respect to noise and distortions, invariance under different transformation etc., are taken care of in the proposed algorithm. For our problem, in-order to take care of skew and rotational invariance, to a little extent, a small survey is done and came out with the idea of splitting the whole contour into two half's namely upper contour and lower contour. The various methods that are worked out to split the contour are Mean of medians, Median of medians, Mode of medians and least square fit of medians.

For all the above methods, the median points of the whole contour are to be found. One method for getting the median points is finding the average values of y - minimum and y - maximum for each x ranging all along the length of the contour

2.5.1 Mean of Median

In this method all the median values are taken into consideration to find the mean value This is obtained by dividing the summation of all the median values by its frequency Truncation function is applied on the mean and the value so obtained is used to find the extreme intersection points of the contour. Once these extreme points are found out, the lower and upper contours are obtained by simply traversing the contour in anti-clock wise direction (Traversing from one extreme intersection point to the other extreme point and again from there to the first extreme point). It is straight forward method and gave good results when there is no skew in words. But if the size of the characters varies a lot within a word, this method fails to find the median line that can cover all the characters. Also if the skew of the word is more, this method fails.

2.5.2 Median of Medians

In this method, all the median values are taken into consideration to find the median value. This is obtained by first sorting all the median values and then taking the middle one, if the frequency is odd, and the average of the middle ones, if the frequency is even. Truncation function is applied on the median and the value so obtained is used to find the extreme intersection points of the contour. Upper and lower contours are obtained by simply traversing the contour in anti clock wise direction This gives a line that is very similar to mean of median's line. It has all the advantages and disadvantages of that of mean of median's line

2.5.3 Mode of Medians

In this method all the median values are taken into consideration to find the mode value This is obtained by finding that value that is repeated maximum number of times. If two or more numbers repeat the maximum number of times a tie occurs and the average of all these numbers is taken as the mode value. Truncation function is applied on the mode and the value so obtained is used to find the extreme intersection points of the contour. Traversing the contour in anti clockwise direction from one extreme intersection point to the other extreme point and again from there to the first extreme point gives upper and lower contours. This gives a line that is not as good as a mean of median's line or a median of median's line.

2.5.4 Least square fit of medians

The other method that is commonly used is, to fit a straight line through a set of data using a least mean square approach. Here in our problem the data is a set of median values. The basic idea is to fit a reference line through these set of points so that the sum of the squares of the distance of each point to the line is minimized. The true distance of a point is the length of the normal vector joining the line to the point. The extreme intersection points of the contour with this line are found and traversing the contour in anti clockwise direction produces lower as well as upper contours

The equation of the line is represented as $y = ax + b$ The vertical distance between the line and the actual data points is computed first. In-order to find the best line that fits our data, the sum of these distances should be minimized This sum is called the error sum and is computed as follows

$$E = \sum_{k-1}^{n} [y_k - (ax_k + b)]^2$$

The minimum occurs when the derivation equals zero. Therefore the partial derivatives of E with respect to the two unknown coefficients **a** and **b** are set to zero. This gives us the two equations :

$$\frac{\partial E}{\partial a} = -2\sum_{k=1}^{n}[y_k - (ax_k + b)]x_k$$

$$\frac{\partial E}{\partial b} = -2\sum_{k=1}^{n}[y_k - (ax_k + b)]$$

Solving these two equations for **a** and **b** gives :

$$a = \frac{\sum_{k=1}^{n}[(x_k - x)(y_k - y)]}{\sum_{k=1}^{n}(x_k - x)^2}$$

$$b = y - a\,x$$

Once the slope **a**, and the y intercept **b** has been calculated the equation of the straight line that best fits the data has been determined

This method of finding a least mean square fit line for a contour has produced very good results and is superior of all the above It works well for words that are tilted as well as words with characters of different size A set of examples will demonstrate the superiority of this method

2.5.5 Examples

2.5.5.1 Word with no skew

Here in this section, the images of Least mean square fit line, Mean of medians line, Median of medians line, Mode of medians line for a word with no skew are given and is as follows.

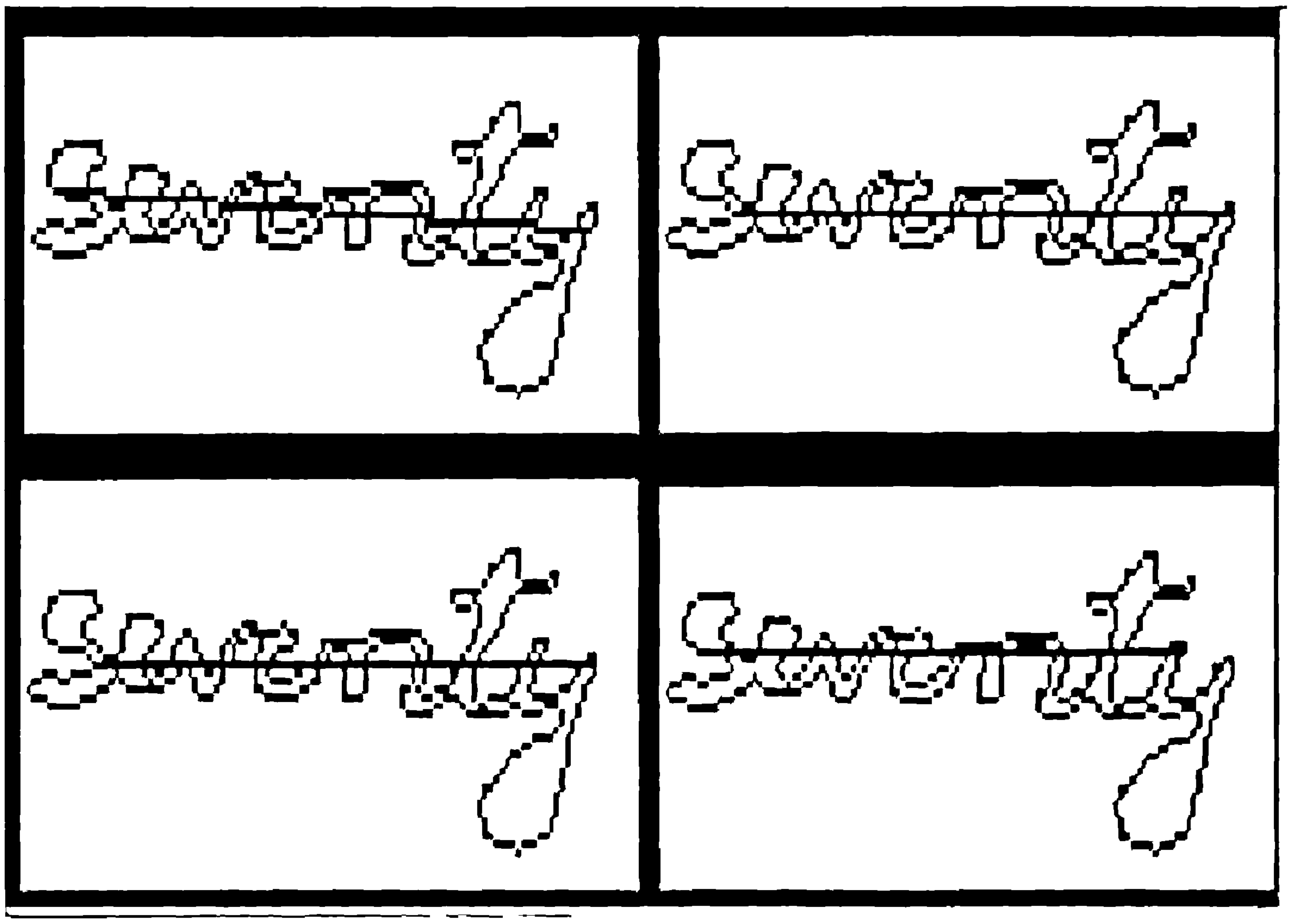


Figure 2.7 : Least mean square fit line, Mean of medians line, Mode of medians line and Median of medians line (Clockwise from top left, Resolution 320 x 200)

2.5.5.2 Word skewed upwards

Here in this section, the images of Least mean square fit line, Mean of medians line, Median of medians line, Mode of medians line for a word that is slanted upwards are given and is as follows.

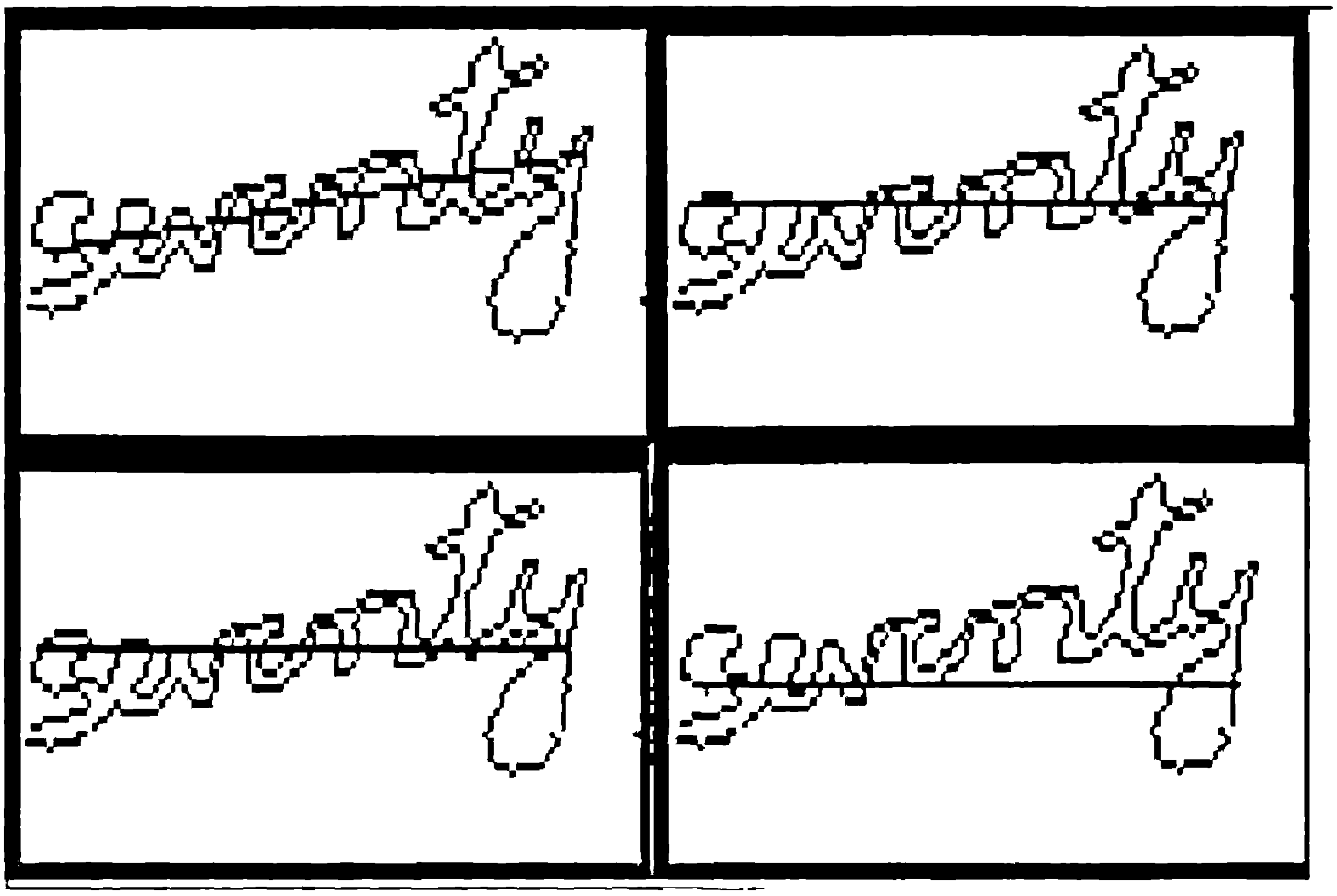


Figure 2.8 : Least mean square fit line, Mean of medians line, Mode of medians line and Median of medians line (Clockwise from top left, Resolution 320 x 200)

2.5.5.3 Word skewed downwards

Here in this section, the images of Least mean square fit line, Mean of medians line, Median of medians line, Mode of medians line for a word that is skewed downwards are given and is as follows.

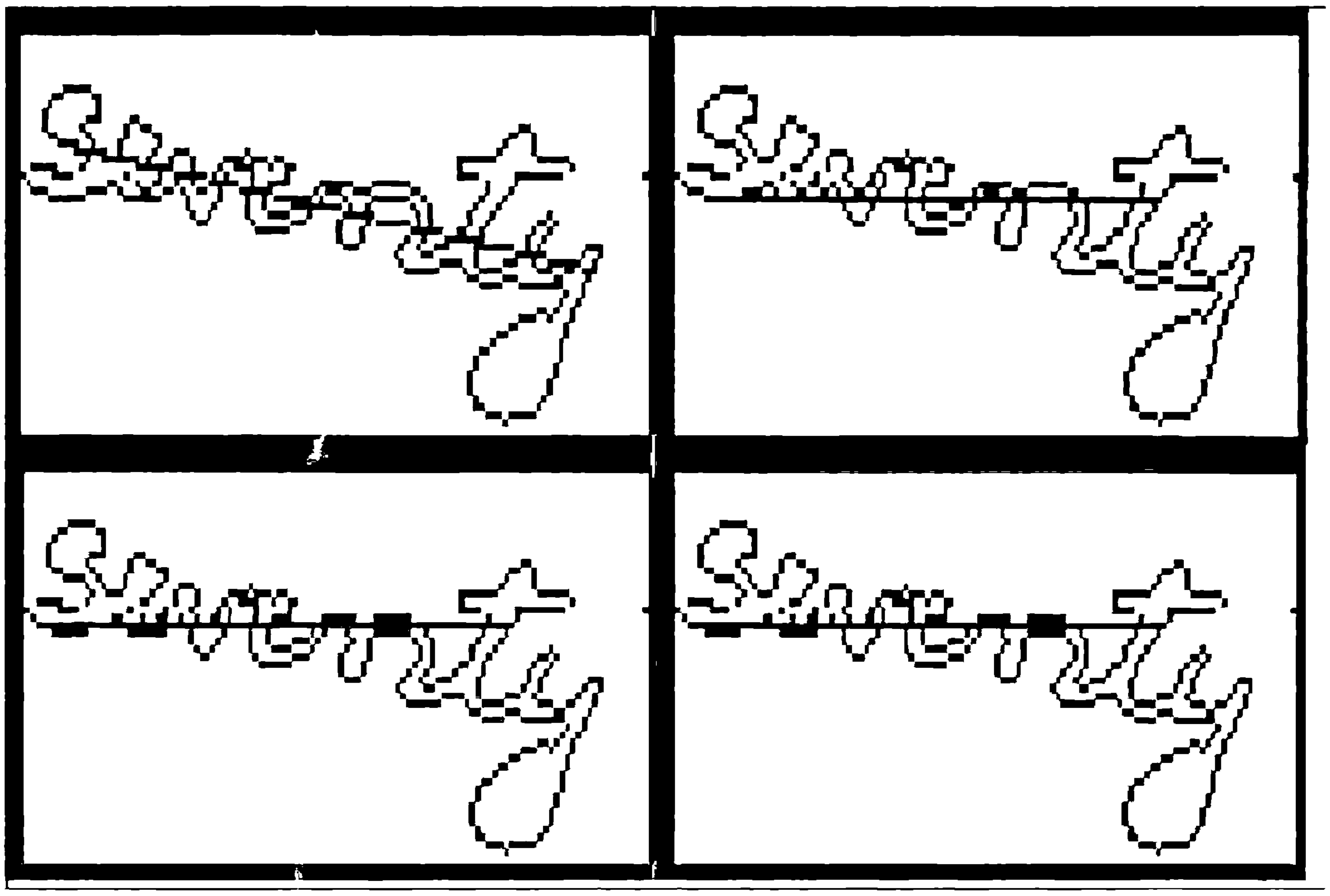


Figure 2.9 : Least mean square fit line, Mean of medians line, Mode of medians line and Median of medians line (Clockwise from top left, Resolution 320 x 200)

2.5.5.4 Word with characters of different size

Here in this section, the images of Least mean square fit line, Mean of medians line, Median of medians line, Mode of medians line for a word with characters of different size are given and is as follows.

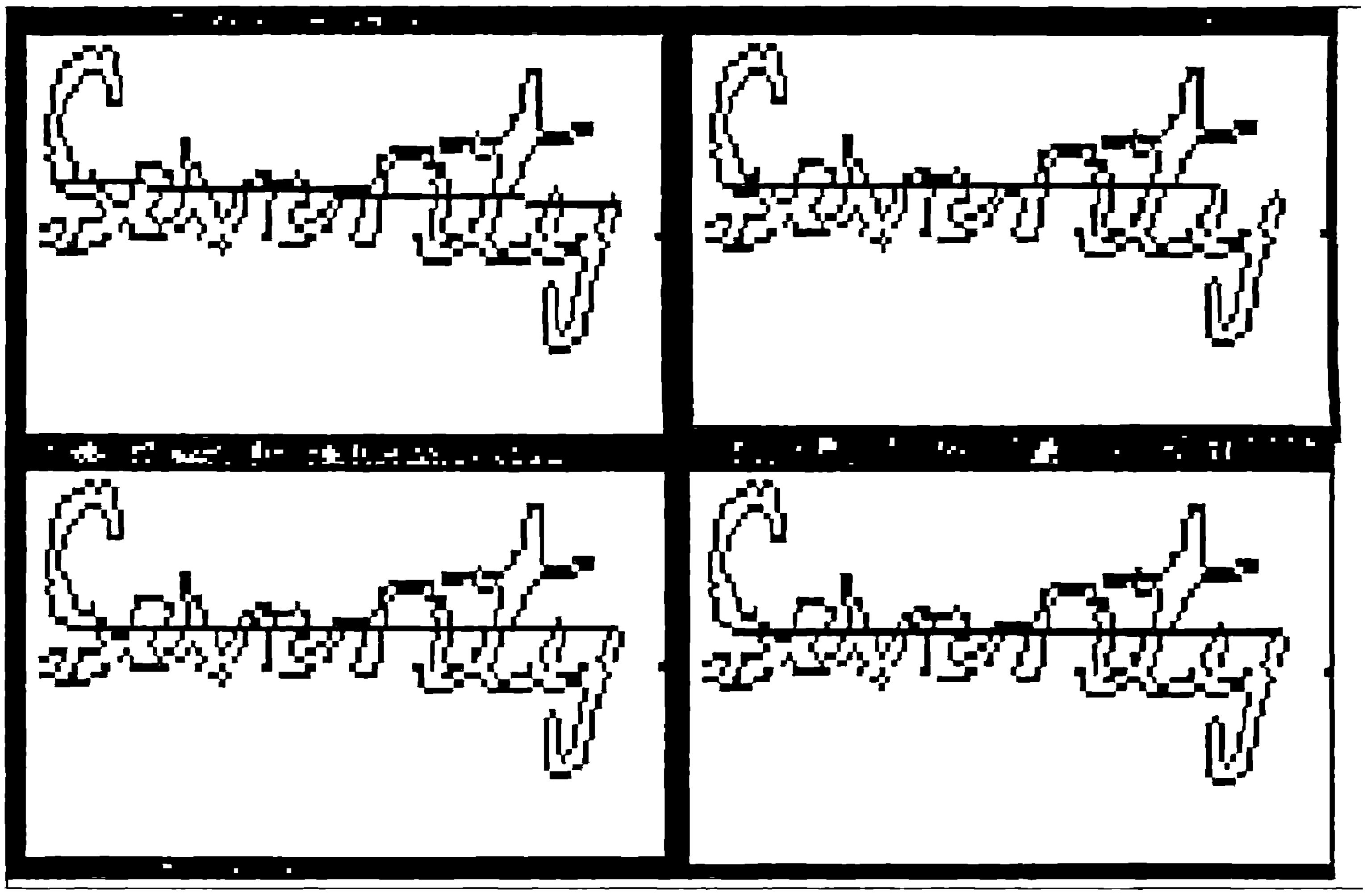


Figure 2.10 : Least mean square fit line, Mean of medians line, Mode of medians line and Median of medians line (Clockwise from top left, Resolution 320 x 200)

From all the above examples it is observed that the least mean square fit line is giving the best possible median line over other methods.

Chapter III

Edit Distance Computation

String matching based methods such as edit distance computation are conceptually very simple and easy to implement. They have low computational complexity and are amenable to parallelization and VLSI implementation [Bunke & Bühler:93],[Maes:91],[Dinstein et al.:91]. The comparison of string is achieved using three standard elementary edit operations namely Insertion Deletion and Substitution. To achieve the similarity between strings using edit distance method, one usually assigns a distance for these elementary symbol operations. Then the inter-pattern distance is computed as a function of these elementary symbol edit distances. Recognition using distance criteria is obtained by essentially evaluating the string in the dictionary that is "closest" to the input unknown word.

3.1 String Matching using Edit Distance (Levenshtien's)

If A is an alphabet of symbols and $X = x_1,x_2 ...... x_n \in A^*$, $Y = y_1,y_2 ...... y_m \in A^*$. The distance between X and Y is defined in terms of elementary edit operations that are required in order to transform X into Y. Three different types of edit operations are considered :

(a.) Substitution of a symbol $a \in A$ in X by symbol $b \in A$ in Y, $a \neq b$;

$\mathbf{d_s(.,.)}$ is a map from $\mathbf{A \times A \rightarrow R^+}$ and is the substitution map. $\mathbf{d_s(a,a)=0}$.

(b.) Insertion of a symbol $\mathbf{a \in A}$ in $\mathbf{Y}$;

$\mathbf{d_i(.)}$ is a map from $\mathbf{A \rightarrow R^+}$ and is called the insertion map.

(c.) Deletion of a symbol $\mathbf{a \in A}$ in $\mathbf{X}$;

$\mathbf{d_e(.)}$ is a map from $\mathbf{A \rightarrow R^+}$ and is called the deletion or erasure map.

Symbolically $\mathbf{a \rightarrow b}$ is for Substitution, $\in \rightarrow \mathbf{a}$ for an Insertion and $\mathbf{a} \rightarrow \in$ for an Deletion. Obviously any given string **X** can be transformed into any other sequence of symbols **Y** by repeated application of these edit operations.

Given two strings **X** and **Y**, there are usually more than one sequence of edit operations transforming **X** into **Y**. The edit operations are used for modeling variations that may change an ideal prototype string into its actual, noisy version. Depending on the particular application, certain distortions, i.e., edit operations, may be more likely than others. In order to take account of this observation, costs of edit operations are introduced. Small costs are defined for distortions that occur frequently and high costs for infrequent distortions. Symbolically $\mathbf{C(a \rightarrow b)}$ is for the cost of Substitution $\mathbf{a \rightarrow b}$, $\mathbf{C(\in \rightarrow a)}$ is for the cost of Insertion $\in \rightarrow \mathbf{a}$ and $\mathbf{C(a \rightarrow \in)}$ is for the cost of the Deletion $\mathbf{a} \rightarrow \in$. All costs are assumed to be real numbers greater than or equal to zero.

If $\forall$ **a, b** $\in$ A the distance is assigned unit values as below :

$$\mathbf{d_s(a,b)} \quad = \mathbf{1} \text{ if } \mathbf{a \neq b}$$

$$= \mathbf{0} \text{ if } \mathbf{a = b}$$

$$\mathbf{d_i(a) = d_e(a) = 1 \ \forall \ a}$$

the inter-string edit distance is called the levenshtien distance.

Now if $s=e_1,e_2,\ldots\ldots.e_k$ be a sequence of edit operations transforming string **x** into string **y**. Then the cost of this sequence is defined as $C(s)=\sum_{i=1}^{k} C(e_i)$.

Given two strings **x** and **y**, the distance **d(x,y)** between **x** and **y** is given by $d(x,y)=\min\{C(s) | s$ is the sequence of edit operations transforming **x** into **y**$\}$.

3.2 Method for finding edit distance(Bunke's)

In this section Bunke's method [Bunke & Bühler:93]of computing edit distance is discussed .

This method differs from other string matching based approaches to 2D shape recognition in a number of ways. First it uses pixel based representation that does not require the segmentation of the boundary of a shape into curves of some type before matching. On the other hand, this results in longer strings and more time being consumed by the matching algorithm. On the other hand the computational time needed for preprocessing is greatly reduced and the problem of missing and extraneous boundary segments are avoided. The algorithm can handle translation rotation and scaling of arbitrary 2D shapes.

The symbolic representation of both the prototype and an unknown shape is given as a sequence of numbers $x=(x_1,x_2,\ldots.x_n)$ where each x_i is an angle $0<x_i<2\Pi$. Here all the angles are quantified. The basic algorithm used for string matching is that of Wagner and Fischer. The cost functions for the basic edit operations are assigned as follows.

$d_s(a, b) = |a-b|$ or $|a-b|^2$ if $a \neq b$

0 if a = b

d_i (a) = d_e (a) - constant (depending on the likelihood of the corresponding edit operation)

all arithmetic operations on these angular values are to be taken modulo 2Π

By slightly modifying the matching algorithm proposed by Wagner and Fisher[Wagner & Fischer:74] rotational invariant and scaled invariant matching algorithm can be obtained. But the problem only lies in how you are going to represent the image and the stick length you are going to take into consideration. The algorithm that is employed for the present work is very similar to this and can handle translation as well as rotational invariance. In our work scaling is done in the pre processing stage itself. The difference lies in the representation and the approach.

3.3 Optimal syntactic pattern recognition using edit distance

In this section Oommen's method [Oommen & Kashyap:96]of computing edit distance is discussed in detail.

This technique de-coupled the occurrence of noisy observations into two distinct modules. The first of these formally describes the module that accounts for the dictionary of ungarbled observations and the second describes the channel that syntactically corrupts error free signals. Here the transmitted symbols can only be substituted for or deleted, and received symbols are obtained as either a result of transmitted symbols being substituted for or as inserted symbols. Here the dictionary used is finite and is even small.

In this the channel is considered as an excited random string generator whose input is a string **U** and whose output is the random string **Y**. The model for the channel is that **Y** is obtained by mutating the input string with an arbitrary sequence of string deforming operations (substitution, deletion and insertion). There are two special symbols used here i.e. λ and ξ (output and input null symbols). Transforming symbol $\mathbf{a} \in \mathbf{A}$ to λ is termed as deletion of symbol **a**; and transforming symbol ξ with some symbol $\mathbf{b} \in \mathbf{A}$ is termed as insertion of **b**. $\mathbf{C_I}$ and $\mathbf{C_O}$ are called as input and output compression operators. $\mathbf{C_I}$ maps $(\mathbf{A} \cup \{\lambda\})^*$ to $\mathbf{A}^*$ and $\mathbf{C_O}$ maps $(\mathbf{A} \cup \{\xi\})^*$ to $\mathbf{A}^*$. $\Gamma(\mathbf{U}, \mathbf{Y})$ is the set of all possible edit operations.

$\Gamma(U, Y) = \{(U',Y') \mid (U',Y') \in (A \cup \{\xi\})^* \times (A \cup \{\lambda\})^*$, and each (U',Y') obey rules (i) -(iii)$\}$

(i) $C_I(U')=U$; $C_O(Y')=Y$

(ii) $|U'| = |Y'|$

(iii) For all $1 \leq i \leq |U'|$, it is not the case that $u'_i = \xi$ and $y'_i = \lambda$.

The meaning of any pair $(\mathbf{U'},\mathbf{Y'}) \in \Gamma(\mathbf{U},\mathbf{Y})$ is that it can be seen to represent one way of editing **U** into **Y**, using the edit operations of Substitution, Deletion, Insertion and transposition.

The next major step is the string generation process. This process describes how a string **Y** is generated given an input string $\mathbf{U} \in \mathbf{A}^*$. There are three distributions defined, namely **G**, **Q** and **S**. **G** is called the Quantified insertion distribution and **Z** is a random variable i.e. the number of insertions that are performed in the mutating process. The quantity $\mathbf{G}(\mathbf{z} \mid \mathbf{U})$ is the probability that the number of insertions is **z** given that **U** is the input word. **G** has to satisfy the constraint $\sum_{z>0} G(z/u) = 1$. **Q** is called the Qualified insertion distribution. The quantity **Q(a)** is the probability that the symbol **a**∈**A** will be inserted symbol conditioned on the fact that an insertion operation is to be performed. **Q** has to satisfy the constraint $\sum_{a \in A} Q(a) = 1$. **S** is called the Substitution and deletion distribution . The quantity **S(b | a)** is the conditional probability that the given symbol **a**∈**A** in the input string is mutated by a stochastic substitution or deletion in which case it will be transformed into a symbol **b** ∈ (**A**∪{**λ**}). If **b** = **λ** then it is conditional probability of **a** being deleted else it is conditional probability of **a** being substituted by **b**. S has to satisfy the constraint $\sum_{b \in (A \cup \{\lambda\})} S(b/a) = 1$

Algorithm Generate_ string
input : The word U and the distributions G, Q, S
output : A random string Y which garbles U with substitution, insertion and deletion mutations
Method :

1. *Using G randomly determine z, the number of symbols to be inserted in U.*
2. *Randomly generate an input edit sequence $U' \in (A \cup \{\xi\})^*$ by randomly determining the positions of the insertions among the individual symbols of U.*
3. *Randomly & independently transform the occurrence of ξ into symbols of A using Q.*
4. *Randomly & independently substitute or delete the non-inserted symbols in U' using S.*

The next step is to evaluate probabilities : Pr[Y | U].

Algorithm : Evaluate_probabilities

Input : The string U, Y and the distribution G, Q and S.
Output : The array W(i,e,s)for all permissible values of i,e and s and the probability Pr[Y/U]
Data structures used : W() is a three dimensional matrix to store the intermediate values during the distance computation. M is length of Y and N is length of U.

```
Method :
$W(0,0,0) = 1$; $Pr[Y/U] = 0$;
For $i = 1$ to $M$ Do $W(i,0,0) = W(i-1,0,0)*Q(y_i)$
For $e=1$ to $N$ Do $W(o,e,0)=W(0,e-1,0)*S(\lambda/u_e)$
For $s=1$ to $R$ Do $W(0,0,s)=W(0,0,s-1)*S(y_s/u_s)$
For $i=1$ to $M$ Do
        For $e=1$ to $N$ Do
                $W(i,e,s)=W(i-1,e,0)*Q(y_i) + W(i,e-1,0)*S(\lambda/u_e)$
For $i=1$ to $M$ Do
        For $s=1$ to $M-i$ Do
                $W(i,0,s)=W(i-1,0,s)*Q(y_{i+s}) + W(i,0,s-1)*S(y_{i+s}/u_s)$
For $e=1$ to $N$ Do
        For $s=1$ to $N-e$ Do
                $W(0,e,s)=W(0,e-1,s)*S(\lambda/u_{s+e}) + W(0,e,s-1)*S(y_s/u_{s+e})$
For $i=1$ to $M$ Do
        For $e=1$ to $N$ Do
                For $s=1$ to $Min[(M-i),(N-e)]$ Do
                $W(i,e,s)=W(i-1,e,s)*Q(y_{i+s})+W(i,e-1,s)*S(\lambda/u_{e+s})+W(i,e,s-1)*S(y_{i+s}/u_{e+s})$
For $I = Max[0,M-N]$ to $M$ Do
        $Pr[Y/U] = Pr[Y/U]+G(i)*(N!i!)/(N+i)!*W(i,N-M+i,M-i)$
```

Using this model, it is easy to see how optimal syntactic pattern recognition can be obtained. Indeed if the distributions **G**, **Q**, **S** are known pattern recognition can be achieved by evaluating the string $\mathbf{U}^*$ which maximizes the probability **Pr[Y | U]** over all **U** in the dictionary.

Chapter IV

Proposed method for String matching and Implementation details

The main aim of this project is to recognize a word as a whole, or a set of hypothesized words (i.e., numerals one to twenty, multiples of tens, hundred and thousand), given a contour representation. In the last Chapter various works related to edit distance computation in the frame work of string matching and dictionary for the contour representation of a binary image are discussed.

Here in this Chapter a detailed description of what all is employed at various stages, for achieving the goal is discussed in detail.

1. Preprocessing techniques employed shall be discussed.
2. The method used for extracting the contour for the normalized binary image and the proposed methods for string matching using edit distances.
3. Algorithms along with their description as well as the data structures used shall be discussed.

The input is a gray level image with 8-bits per pixel. The scanning of images is done with proper resolution setting from a scanner.

4.1 Pre-processing

The pre-processing as described in Chapter Two has mainly 4 phases. They include

1. Extracting bi-level image from Gray level image
2. Normalization and Filtering
3. Extracting contour from bi-level image and
4. Splitting of contour into upper and lower contours

Here the following sections give the brief description and implementation details of each of the above mentioned pre-processing techniques. A running example shall be take into consideration through out this Chapter.

4.1.1 Extracting Bi-level image:

The gray level image that is taken as input is a TIFF file. The information that is present in a gray level TIFF image is discussed in detail in Appendix A. A small routine was written that reads a gray TIFF image and converts it into Bi-level image. The algorithm is as follows :

```
Algorithm Tiff_read( )

Input : A gray level TIFF image file
Output : A Binary image file i.e. stored in img.bin file  along with Width and Height.
Data structures : Tiff_header
structure {
        int lsb;
        long imageht;              : image height
        long imagewidth;           : image width
        long bitsps;               : Bits per sample
        long stripoffset,            . Strip offset
        long rowsps,                 : Rows per strip
        int resolution,              : Resolution
      }Tiff header;
```

The following three unions are used to put the bytes from the header into an integer

```
union short_char {
        short int s_num;
        char  s_word[2];
                };

union int_char {
        int  i_num;
        char i_word[2];
                };

union long_char {
        long  l_num;
        char  l_word[4];
                };

integer thresh = 180
unsigned character temp[64];

Method  tiff_read()
{
function : unsigned int  ext_int_temp();
function : unsigned long ext_long_temp();

Step 1 : Open Tiff file in read binary mode;
Step 2 : Read into temp the first  8 bytes of image file;
Step 3:Determine if the file uses MSB first or LSB first by checking temp[0] and accordingly set lsb to 1 or 0 respectively
Step 4 : Get the offset of ifd from temp (i.e., 4 bytes) Take care if it is LSB first and set notend flag to TRUE
Step 5 :While(notend is TRUE) Do steps 6 to 8
Step 6 : Seek to the ifd and read the number of entries in the ifd
Step 7 : For (all the entries) Do Get the tag type and depending on the tag type get the information for Subfile type, Image width, Image length, Bits per sample, Strip offset,Rows per strip and Resolution
Step 8 : Get the offset to next ifd. If offset is 0 then set notend flag to FALSE
```

Step 9 : Seek to the starting point of the image data

Step 10 : Read the image data into a single dimensional array fil

Step 11 : For (each i in fil) do if (fil[i] > thresh) then enter the current pixel value as '0'(white) in bi-level image file i.e. img.bin. else enter the current pixel value as '1' (black)in bi-level image file. also enter the image height and width information.

function : unsigned integer ext_int_temp()
input : char array temp,integer lsb,integer start_point;
output : short character sc;
Description : Converts the content in temp into short character starting at start_point and depending on lsb.

function : unsigned long ext_long_temp()
input : char array temp,integer lsb,integer start_point;
output : long character lc;
Description : Converts the content in temp into longcharacter starting at start_point and depending on lsb.

The binary image that is obtained from a gray level image by using the above mentioned algorithm depends mainly on the threshold value taken. If the threshold value is very small then there is a chance that some of the information is lost. And if the threshold value is very large, some unwanted information is incorporated into the image. So a proper threshold value has to be taken into consideration. This requires some experimentation with the input data. A threshold value of 180 for a 256 gray level image is obtained by doing some experimentation on the data set.

4.1.2 Normalization and Filtering

The next step that is done after getting a binary image by proper thresholding is normalization of image. The image is now condensed or enhanced depending on its initial size. An average height and width of 50 X 112 is obtained by performing a small survey on all the images that are used as a data set. So this dimensions are taken as a standard and all the images are resized to this dimensions

After getting the normalized binary image the image is read into a single dimensional array i.e. img[]. Before using this image for recognition, noise need to be removed. The presence of noise and the need for removing it is discussed in detail in Chapter Two. The technique that is employed for the present work to filter out the noise is based on the following principle.

Noise can be easily identified by computing the size of connected components. Single pixel components are the first to be removed. Second, components that have very small size with respect to the average width and height of the image are also removed. The second stage of noise removal is done at the stage of constructing the contour. One disadvantage of the above method is that i-dots and j-dots are also removed. But for our problem, only the upper and lower contours are taken into consideration for string matching. Continuity in the contour should be bothered about, so the above stated disadvantage is not a problem. To get a continuous contour it is assumed that the input word is fully connected. If this is not the case then there is a chance for portions of the word to be lost. The algorithm is as follows :

```
Algorithm Filter( )
Input : Binary image stored in an array img[ ] along with height and width.
Output : Binary image with all the noise removed.
Method :
Step 1 : Read the Binary image img[ ],width and height.
Step 2 : Start at the left top corner and repeat Steps 3 and 4 until all the image is processed first left to right then top to bottom.
```

Step 3 : If the current pixel is black and it has no connected components then the current pixel color is replaced with background color and go to Step 2.
Step 4: If the current pixel is black and has a connected component then simply go to Step 2.

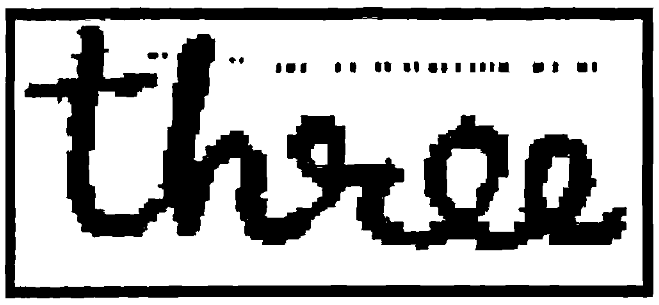


Figure 4.1 : Binary image before normalization with noise (Twice the actual resolution)

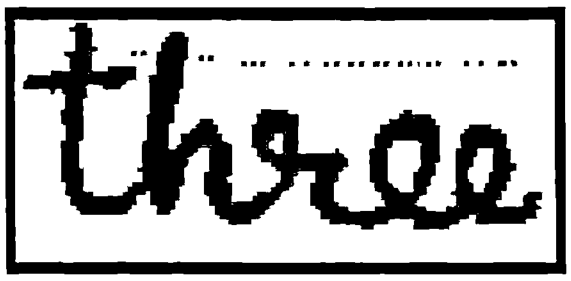


Figure 4.2 : Binary image after normalization with noise (Twice the actual resolution)

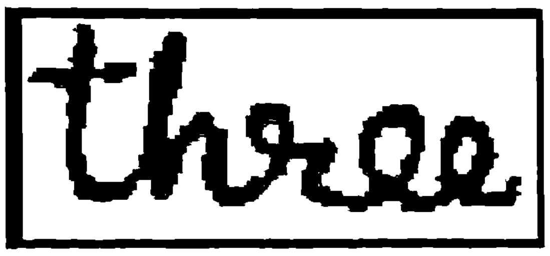


Figure 4.3 : Normalized binary image with noise removed (Twice the actual resolution)

4.1.3 Extracting Contour

After getting a normalized binary image with all the noise removed our next step is to give this image as input to a contour generation function. The algorithm that is used in generating the contour is by Pavlidis[Pavlidis:82] with a small modification. The modification is done to incorporate noise removal at a second stage (i.e. as discussed in the previous section). The input to this is a normalized binary image along with width and height values, and the output is a linked list with the x-y coordinates along with the chain code of the external contour. The basis behind selecting chain code representation is given in Chapter Two. The algorithm that is used to extract the external contour is as follows :

```
Algorithm : Trace_contour( )
Input : A single dimensional array img[ ], width and height of the image and a starting point of the contour.
Output : A linkedlist with 4 fields : x-coordinate,y-coordinate,chaincode,link;
Notation : The set of points R, A the start point in set R, C the current point whose neighborhood is examined, S the search direction, first is flag that is true only when the tracing starts, and found is flag that is true when a next point on the contour is found.
Method :
Step 1 : Set the current point C to A, the search direction S to 6, and the flag first to true. Store C and S into a node in linked list.
Step 2 : While C is different from A or the flag first is true do steps 3 to 10.
Step 3 : Set the flag found to false.
Step 4 : While found is false do steps 5 to 9 , atmost three times.
Step 5 : If B, the (S-1)-neighbor of C, is in  R  then
Step 6 : Set  C to B, S to S-2, and found to true. Store C and S in Linked list
Step 7 : Else if B, the S neighbor of C, is in R, then Set C to B, and found to true.
Step 8 : Else if B, the (S+1)-neighbor of C, is in R, then set C to B and found to true.
Step 9 : Else increment S by 2
Step 10 : Set first to false
```

Step 11 If the length of the current contour is less than 1/10 th of the average image height or width, this contour is discarded and again start the whole process with a new starting point.

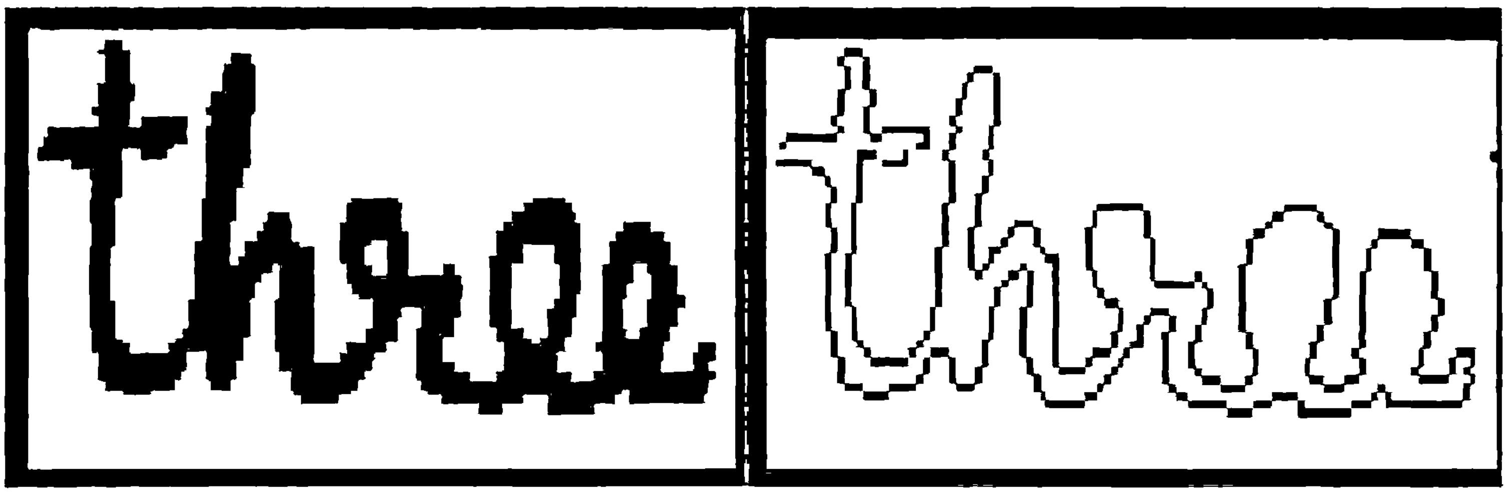


Figure 4.4 : Binary image and external contour of it (left to right, Resolution 320x200)

```
60000070776566766566666666766766700700010001121007667001212212221
21101766656666676767700000101111110221100765666667707000007000100
100076000000001001176000000000001213124456544443121212123223223434
444556656666666665765454434321212122122222223334444545545665666666
67676655444344332212121233544444322223223444444565666676770766565
44554443222222222222433234455546553222212221212222212222234455656
57566666665666656666665665555444443332223222221222221222210070001
1022444444454322123222344576656657544444455
```

Figure 4.5 : 8-direction chain code of the contour in figure 4.4 with actual resolution

4.1.4 Splitting of contour

The output of the contour generation algorithm is a linked list with x-y coordinates along with chain code. The usual step after obtaining binary image is to compare the whole contour with all the contour's in the data set. But this sort of comparing the whole contour with all the contours in the data set consumes a lot of space as well as time. So it is proposed here that by splitting the contour into two halves namely upper and lower contours the space and time consumed will be considerably reduced. There are many algorithms available for splitting the contour into two halves, some of which are discussed in Chapter Two. By experimenting all these methods on various types of samples it is observed that least mean square fit line generates the most appropriate median line that splits the whole contour into two halves. The method of finding least mean square fit for a set of pixels in an image is based upon linear regression (discussed in Chapter Two).

For our work the data would be a set of median points all along the width of the contour and our aim would be to find a least mean square fit median line for this data. This needs to find the median points all along the length of the contour before finding the least mean square fit line. The method for finding the median points is very simple that as described in Chapter Two is finding the average value of y minimum and y maximum for each values of x all along the length of the contour. The algorithm that is employed in the present work is as follows :

Algorithm : median_points( )
Input : linked list containing the contour, along with width and height of the image
Output : Linked list containing all the median points.
Notation : integer min,max to store the minimum and maximum values of y for each x along the width of the contour.
Method :
Step 1 : For (each x along the width of the contour) do steps 2 - 6
Step 2 : Initialize min to height and max to zero get the starting address of the linked list that contains the contour.
Step 3 : While linked list not equal to NULL) do steps 4-5

```
Step 4 : if(current nodes x value equals x) then if (current nodes y value less than min) then min is assigned the current nodes y value else if (current nodes y value is greater than max) then max is assigned the current nodes y value.
Step 5 : go to the next node in the linked list
Step 6 : enter into a linked list, the current value of x and average value of min and max.
```

After getting the median points, the next step is finding the least mean square fit of all these median points. The algorithm that is employed to obtain least mean square fit line is as follows :

```
Algorithm : Least_square_fit()
Input : Linked lists with all the median points.
Output : The slope and y-intercept of the least mean square fit line.
Notation : Double meanx, meany, meanxdiff, meanydiff, sum_prod_xy_diff, sum_sqr_x_diff,slope,intercept.
Method :
Step 1: Initialize meanx, meany, meanxdiff, meanydiff, sum_prod_xy_diff, sum_sqr_x_diff to zero
Step 2 : For (all median points of the contour) do steps 3 - 4
Step 3 : Find the mean value of x coordinates and assign it to meanx
Step 4 : Find the mean value of y coordinates and assign it to meany
Step 5 : For (all median points of the contour) do steps 6 - 9
Step 6 : meanxdiff is each median point x value minus meanx value
Step 7 : meanydiff is each median point y value minus meany value
Step 8 : sum_prod_xy_diff is the sum of products of meanxdiff and meanydiff which are obtained in step 6 and 7
Step 9 : sum_sqr_x_diff is the sum of square of each meanxdiff obtained in step 6
Step 10 : slope is sum_prod_xy_diff divided by sum_sqr_x_diff
Step 11 : intercept is meany minus slope multiplied by meanx
```

Example of a least mean square fit line for a contour is given in Section 2 5.5

Once the least mean square fit line is obtained, the next step is to find the intersection of this line with the contour. There can be many intersections of the least mean square line with the contour, but only two points are of interest.

(i e., the extreme left intersection and the extreme right intersection). After finding these two extreme points the next step would be to traverse along the contour and get the upper and lower contours. The algorithm for finding the two extreme intersection points is simple and is as follows :

```
Algorithm : find_extremes( )
Input : slope and intercept of the least mean square fit line,Linked list that contains the whole contour along with height of the image.
Output : two extreme points.
Notation : integer min,max,minx,miny,maxx,maxy.
Method :
Step 1 : Initialize min to height and max to zero
Step 2 : while (linked list not equal to NULL) do steps 3 - 4
Step 3 : if (absolute value(current nodes x coordinate minus intercept minus slope multiplied by current nodes y coordinate) is less than 0.005) then if (current nodes x coordinate less than min) then minx is assigned current nodes x value, miny is assigned current nodes y value and min is assigned the value of minx. else if (current nodes x coordinate is greater than max) then maxx is assigned current nodes x value, miny is assigned current nodes y value and max is assigned the value of maxx.
Step 4 : go to the next node in linked list.
```

The two extreme points are stored in (minx, miny) and (maxx, maxy). The next algorithm is the upper and lower contour generator. This takes as input the two extreme points along with the contour and generates as output, the upper contour chain code and lower contour chain code and store them in two different files. The algorithm is as follows :

```
Algorithm upper_lower( )
Input : integer minx,miny,maxx,maxy and a linked list that contains the whole contour.
Output : upper and lower contour stored in two files.
Method :
Step 1 : while (linked list not equal to NULL) do steps 2 8 and 14
Step 2 : if (current nodes x value equals minx and current nodes y value equals miny) then do steps 3 5 6 and 7
Step 3 : while (current nodes x value not equals maxx or current nodes y value not equals maxy) do step 4
```

Step 4 : enter the current nodes chain code into a file that stores lower contour and get the next node in the linked list.

Step 5 : while (linked list not equal to NULL) do enter the current nodes chain code into a file that stores upper contour and get the next node in the linked list.

Step 6 : Start at the first node of the linked list

Step 7 : while (current nodes x value not equals minx or current nodes y value not equals miny) do enter the current nodes chain code into a file that stores upper contour and get the next node in the linked list

Step 8 : if (current nodes x value equals maxx and current nodes y value equals maxy) then do steps 9 11 12 and 13

Step 9 : while (current nodes x value not equals minx or current nodes y value not equals miny) do step 11

Step 10 : enter the current nodes chain code into a file that stores upper contour and get the next node in the linked list.

Step 11 : while (linked list not equal to NULL) do enter the current nodes chain code into a file that stores lower contour and get the next node in the linked list.

Step 12 : start at the first node of the linked list

Step 13 : while (current nodes x value not equals maxx or current nodes y value not equals maxy) do enter the current nodes chain code into a file that stores lower contour and get the next node in the linked list

Step 14 : get the next node in the linked list.

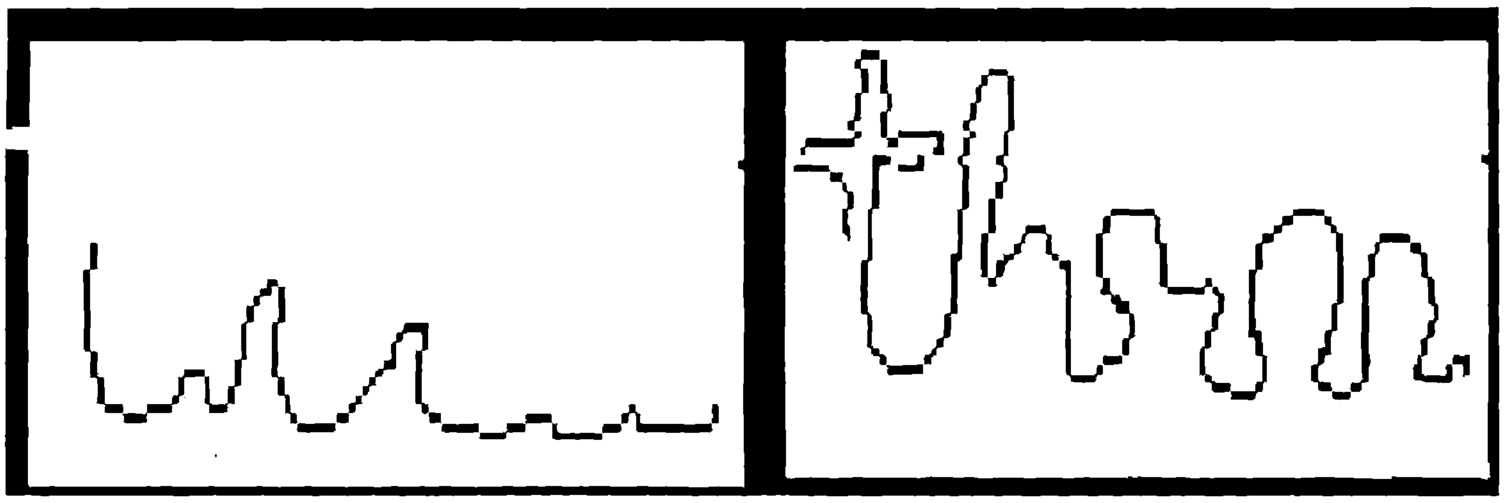


Figure 4.6 Lower and upper contours of the contour shown in fig 4.4

4.2 Matching using edit distances

A number of different algorithms for symbolic string matching have been proposed in the literature some of which were discussed in Chapter Three. This section gives a brief description of the employed algorithm along with its implementation details.

The matching algorithm employed here is based on the edit distance method that is very similar to that of Bunke's[Bunke & Bühler:93]. The idea behind this algorithm is to represent both the unknown sample occurring in the input image, and a set of prototypical models as strings. i.e., sequence of chain codes, and to apply an edit distance matching algorithm to compute the similarity between an unknown input and a prototype. The matching algorithm that is implemented is conceptually simple and easy to implement with low computational complexity. This algorithm computes the measure of similarity, or distance, between two strings in terms of basic edit operations (denoted by **e**). The considered operations allow changing one symbol of a string into another symbol (Substitution), deleting a symbol from a string (Deletion), or inserting a symbol into a string (Insertion). Each operation is assigned a non-negative cost (**C**) The distance is represented by **d**.

The algorithm is as follows :

```
Algorithm : edit_distance( )
Input : string x and string y
Output : edit distance d.
Notation : D is a two dimensional array i.e., $(n-1)$ x $(m-1)$ where n is x and m is y
Method :
Step 1 : initialize D[0,0] to zero
Step 2 : For (i ranging from 1 to n in steps of 1) do $D[i,0] = sum (D[i-1,0], C(x_i \rightarrow \epsilon)$
Step 3 : For (j ranging from 1 to m in steps of 1) do $D[0,j] = sum (D[0,j-1], C(\epsilon \rightarrow y_j)$
Step 4 : For (i ranging from 1 to n in steps of 1) do step 5
Step 5 : for (j ranging from 1 to m in steps of 1 ) do
Step 6 : $m_1$-sum(D[i-1,j-1],C($x_i \rightarrow y_j$)); $m_2$=sum(D[i-1,j],C($x_i \rightarrow \in$)); $m_3$=sum(D[i,j-1],C($\in \rightarrow y_j$)); D[i,j]   min($m_1$,$m_2$,$m_3$)
Step 7 : d   D[n,m]
```

This algorithm has to be run for all the prototypes i.e., all the contours of the data set. One constraint in the above algorithm is that what are the costs assigned to each of the edit operations. It is very simple. All the Deletion costs(C_e) are assigned a value 1 so are Insertion costs(C_i). Substitution cost (C_s) depends on the symbol being replaced by another symbol and is described as follows.

Symbol(x) being replaced	**x Being Substituted by symbols**	**Corresponding Substitution Costs**
0	[0,1,2,3,4,5,6,7]	[0,1,2,3,4,3,2,1]
1	[0,1,2,3,4,5,6,7]	[1,0,1,2,3,4,3,2]
2	[0,1,2,3,4,5,6,7]	[2,1,0,1,2,3,4,3]
3	[0,1,2,3,4,5,6,7]	[3,2,1,0,1,2,3,4]
4	[0,1,2,3,4,5,6,7]	[4,3,2,1,0,1,2,3]
5	[0,1,2,3,4,5,6,7]	[3,4,3,2,1,0,1,2]
6	[0,1,2,3,4,5,6,7]	[2,3,4,3,2,1,0,1]
7	[0,1,2,3,4,5,6,7]	[1,2,3,4,3,2,1,0]

The algorithm that can return the cost depending on the two values **x** and **y**, (i.e., **x** is the symbol being replaced and **y** is the symbol that is substituted in the place of x) is as follows :

```
Algorithm cost( )
Input : Circular linked list,integer i : one being substituted,j : one that is substituted
```

Output Cost for i being substituted by j
Notation Circular linked list is of fixed size and each node in it is assigned a fixed value i.e , [0,1,2,3,4,3,2,1]respectively and is initialized only once at the beginning of the matching process.
Method :
Step 1 : if (i equals 0) then do while (j not equal to 0) get the next node in circular linked list and decrement j by one . finally when j equals 0 return the current nodes value as cost.
Step 2 : else while ((8 minus i) not equal to 0) do get the next node in the circular linked list and increment i by 1
Step 3 : while (j not equal to 0) get the next node in circular linked list and decrement j by one . finally when j equals 0 return the current nodes value as cost.

4.2.1 Upper contour matching

The algorithm that is used for matching the upper contour is the one given in the previous section. In this the string **x** is the upper contour of the input unknown image and y is the upper contour of a known sample in the data set. **n** is the length of unknown sample **x** and **m** is the length of the known sample **y**. The unknown sample **x** has to be matched with all **y** in the data set. After this exhaustive matching only those samples of **y** whose similarity measure is more are selected. One question that has to be answered now is that why is upper contour matching done first rather than matching lower contour first. The answer is that our data set has images that have more number of ascenders than descenders and so performing upper contour matching first seems to give a better matched set. The threshold value of the distance has to be properly set so as to select only those images that are below this threshold value for the next phase of matching. The threshold value should be selected in such a way that not more number of samples are selected and not less number of images are selected for the next phase of matching.

For the running example, after the upper contour matching, only three samples from the dictionary are matched and are the 13 th, 16 th and 20 th variants of the word 'three ' with costs 80, 89 and 86 respectively.

4.2.2 Lower contour matching

This is the second phase of matching. Here only a few images in the data set are left with. Here the matching algorithm used is same as the one used for matching upper contours. In this the string **x** is the lower contour of the input unknown image and **y** is the lower contour of a known sample in the matched data set. **n** is the length of unknown sample **x** and **m** is the length of the known sample **y**. The unknown sample **x** has to be matched with all **y** in the matched data set. Finally only those very few images that have least distance are considered, and proceed to the next phase that is rotational invariance aspect of matching the whole contours.

Fot the running example, from the three samples that are matched in the previous section, the 13 th variant of the word 'three' is matched. So the unknown word is recognized as the word **'THREE'.**

4.2.3 Rotational invariance aspect

At this third stage of matching a very few images in the data set are left with. So now at this stage it is advisable to take into consideration the rotational invariance aspect into consideration. Here the whole contour is taken into consideration while matching. The algorithm that is employed here will take care of rotational invariance and is as follows :

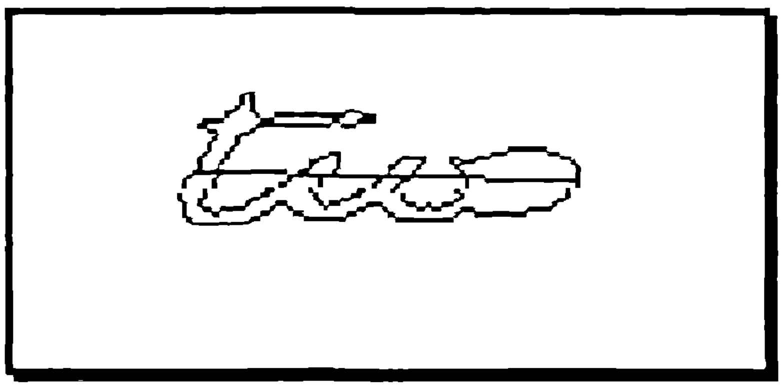
```
Algorithm : cyclic_edit_distance( )
Input : string x and string y
Output : edit distance d.
Notation : D is a two dimensional array i.e., $(n+1) x (m+1)$ where n is x and m is y
Method :
Step 1 : initialize $D[0,0]$ to zero
Step 2 : For ( i ranging from $m+1$ to $2m$ in steps of 1) do $y_i = y_{i-m}$
Step 3 : For (i ranging from 1 to m in steps of 1) do $D[0,i] = 0$
Step 4 : For (i ranging from $m+1$ to $2m$ in steps of 1)do $D[0,i]=\infty$
Step 5 : For (i ranging from 1 to n in steps of 1) do $D[i,0]= sum (D[i-1,0],C(x_i \rightarrow \in)$
Step 6 : For (i ranging from 1 to n in steps of 1) do step 7
Step 7 : For (j ranging from 1 to 2m in steps of 1 ) do
Step 8 : $m_1=sum(D[i-1,j-1],C(x_i \rightarrow y_j))$; $m_2=sum(D[i-1,j],C(x_i \rightarrow \in))$; $m_3=sum(D[i,j-1],C(\in \rightarrow y_j))$; $D[i,j] = min(m_1,m_2,m_3)$
Step 9 : $d = min\{D[n,i]/i=1,.....2m\}$
```

The algorithm not only resolves the tie occurred in phase two but also matches those strings that are the same but are rotational variants of one another. The method is very simple. Here y is repeated to obtain y^2 = yy = $y_1,...y_m,y_1,...y_m$. Now it is sure that y^2 properly contains x as a sub-string independent of the starting position. The matching algorithm by Bunke is changed slightly to set any matrix element **D[0,j], j=1,...,n** equal to 0. This means that we may skip some leading symbols in y^2 without any cost. Further more instead of considering the element in the right lower corner of the matrix **D** we look for an element **D[n,i] i=m,....,2m-1** that is equal to zero. The presence of such an element indicates that y^2 contains x as a substring.

Chapter V

Results

Example 1:

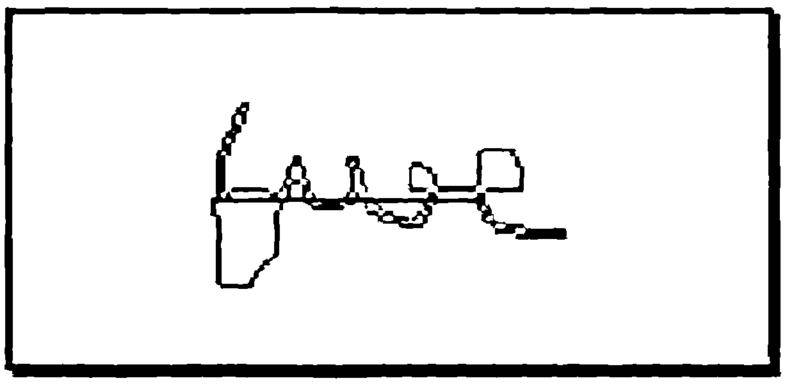


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
3	Two	64
7	Two	56
14	Two	70

Recognized as word 'TWO' after the second phase(Lower Contour Match). Matched sample is **variant 7 of sample two**.

Example 2 :

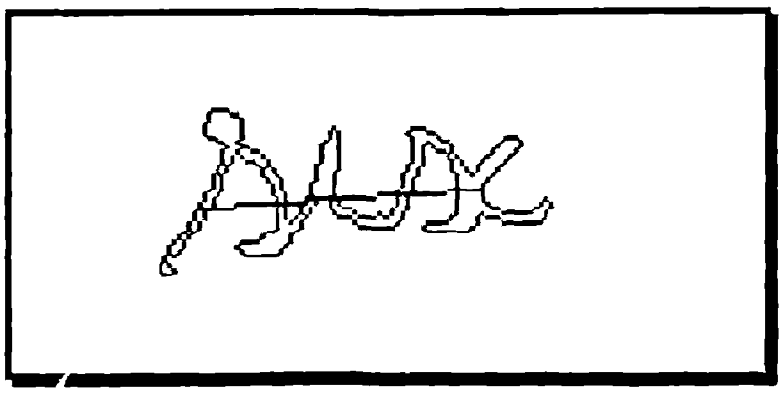


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
15	Five	87
16	Five	99
18	Five	89
23	Five	98

Recognized as word '**FIVE**' after the second phase(Lower Contour Match). Matched sample is **variant 18** of **sample five**.

Example 3 :

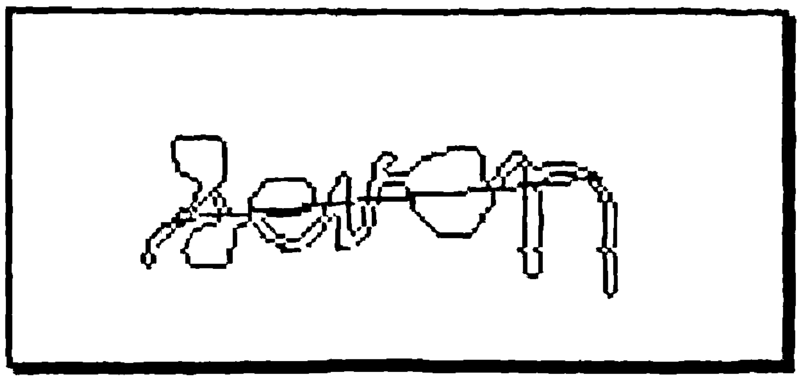


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
6	One	91
2	Six	86
16	Six	100
19	Six	103

Recognized as word '**SIX**' after the second phase(Lower Contour Match) Matched sample is **variant 2** of **sample six**.

Example 4 :



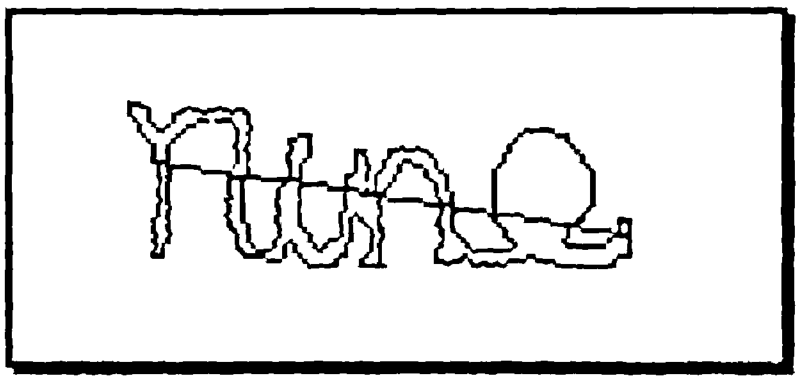
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
21	Five	110
16	Six	101
13	Seven	98
15	Seven	104

Recognized as word 'SEVEN' after the second phase(Lower Contour Match).

Matched sample is **variant 13** of **sample seven.**

Example 5:

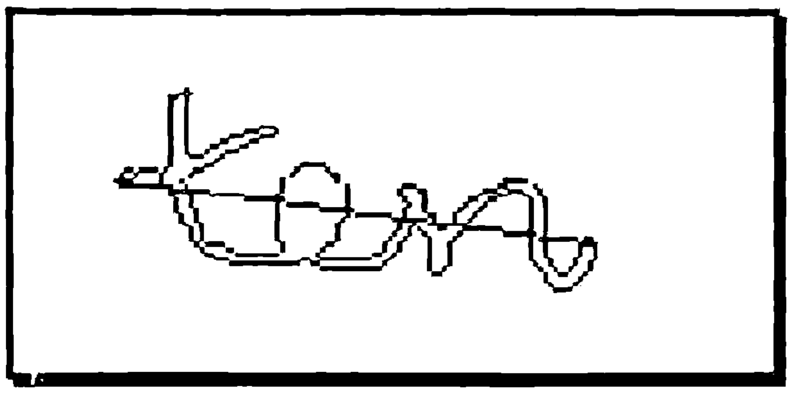


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
10	Nine	140
15	Nine	136
16	Nine	140
18	Nine	140
24	Nine	138

Recognized as word 'NINE' after the second phase(Lower Contour Match). Matched sample is **variant 15** of **sample nine**.

Example 6:

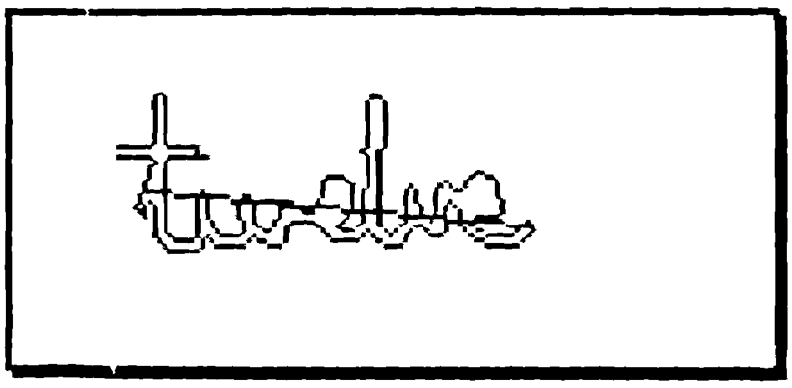


Matched list after the first phase (Upper Contour Match) :

Variant	**Sample**	**Cost**
18	Five	127
7	Ten	127
9	Ten	118
17	Ten	127
20	Ten	116

Recognized as word '**TEN**' after the second phase(Lower Contour Match). Matched sample is **variant 20** of **sample ten**.

Example 7:



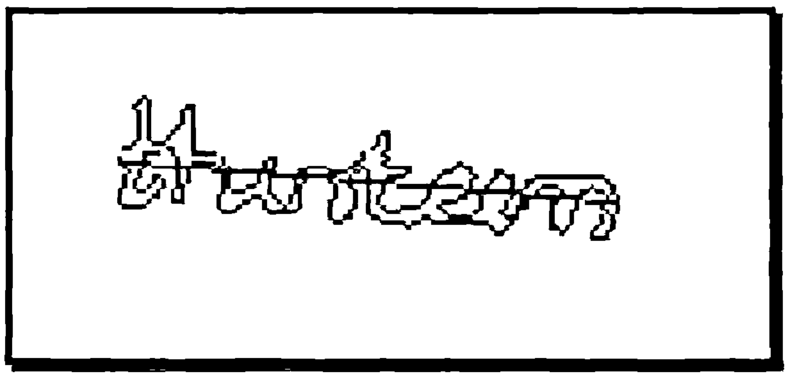
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
9	Twelve	131
12	Twelve	135
16	Twelve	132
17	Twelve	138

Recognized as word 'TWELVE' after the second phase(Lower Contour Match).

Matched sample is **variant 16** of **sample twelve**.

Example 8:



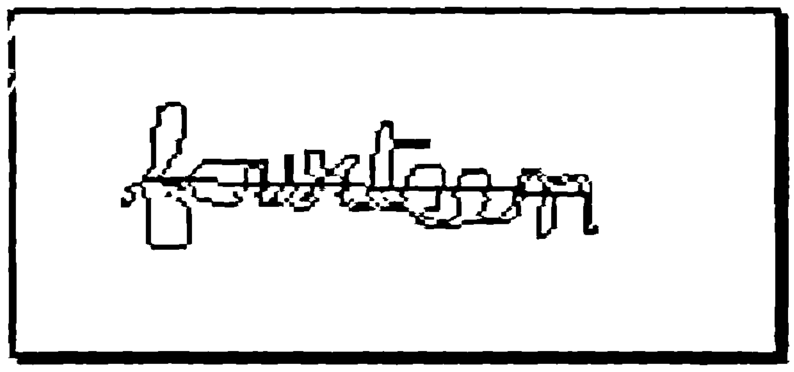
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
3	Thirteen	52
14	Thirteen	61
8	Thirty	54

Recognized as word '**THIRTY**' after the second phase(Lower Contour Match).

Matched sample is **variant 8** of **sample thirty**.

Example 9:

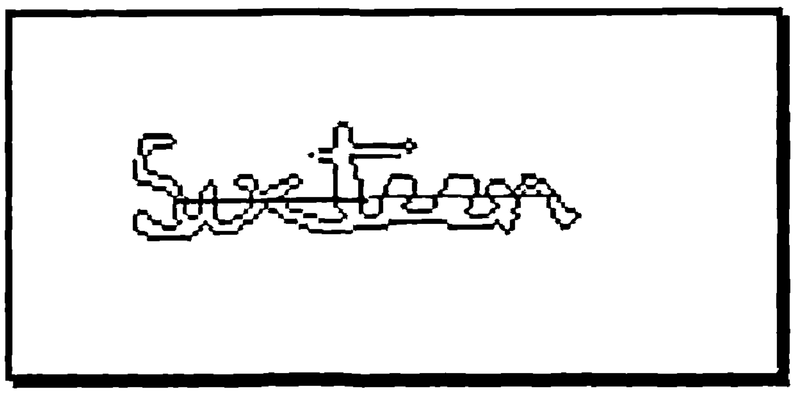


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
11	Thirteen	121
12	Thirteen	119
2	Fourteen	123
6	Fourteen	117
9	Fourteen	121
19	Fourteen	119

Recognized as word 'FOURTEEN' after the second phase(Lower Contour Match). Matched sample is **variant 19** of **sample fourteen.**

Example 10:



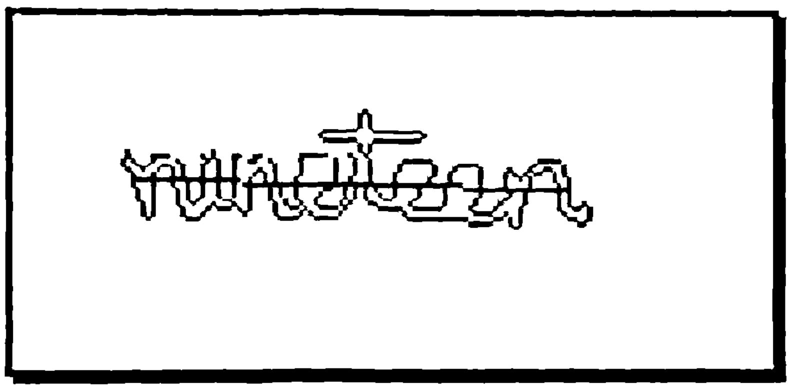
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
3	Sixteen	138
15	Sixteen	122
9	Eighteen	137
11	Nineteen	127

Recognized as word 'SIXTEEN' after the second phase(Lower Contour Match)

Matched sample is **variant 15** of **sample sixteen.**

Example 11:

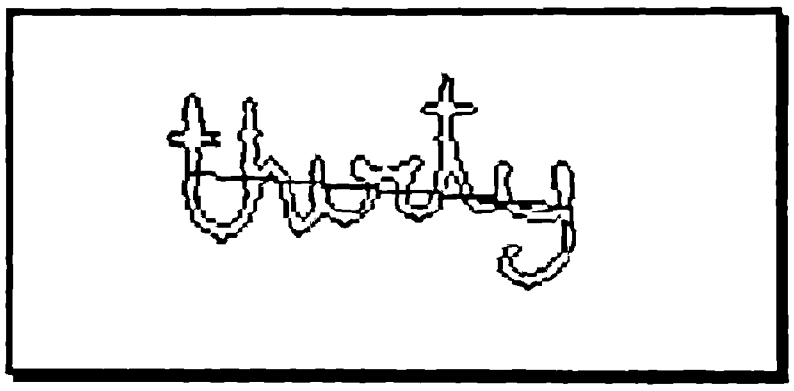


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
15	Thirteen	138
10	Sixteen	128
11	Nineteen	128
12	Nineteen	127
14	Ninety	127
18	Ninety	136

Recognized as word 'NINETY' after the second phase(Lower Contour Match). Matched sample is **variant 14** of **sample ninety**.

Example 12:

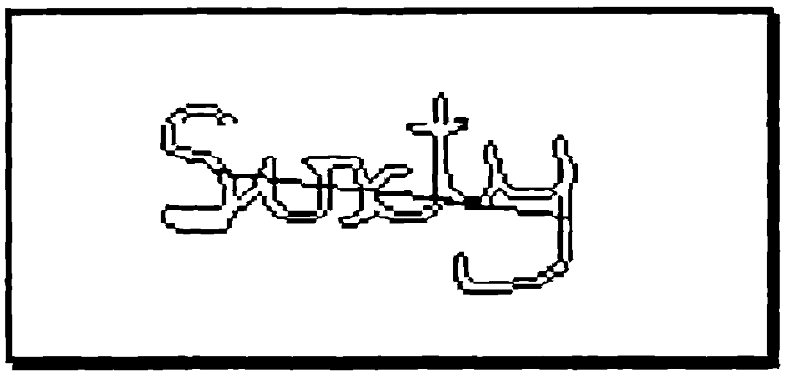


Matched list after the first phase (Upper Contour Match) ·

Variant	Sample	Cost
16	Three	168
13	Thirteen	160
13	Thirty	164
17	Thirty	162

Recognized as word 'THIRTY' after the second phase(Lower Contour Match). Matched sample is **variant 17** of **sample thirty**.

Example 13:

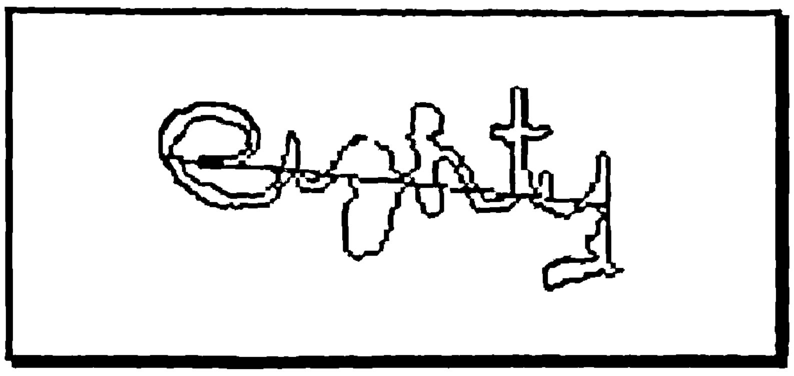


Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
5	Sixteen	155
13	Sixty	155
17	Sixty	160
18	Sixty	158
12	Ninety	161

Recognized as word 'SIXTY' after the second phase(Lower Contour Match). Matched sample is **variant 13** of **sample sixty**.

Example 14:



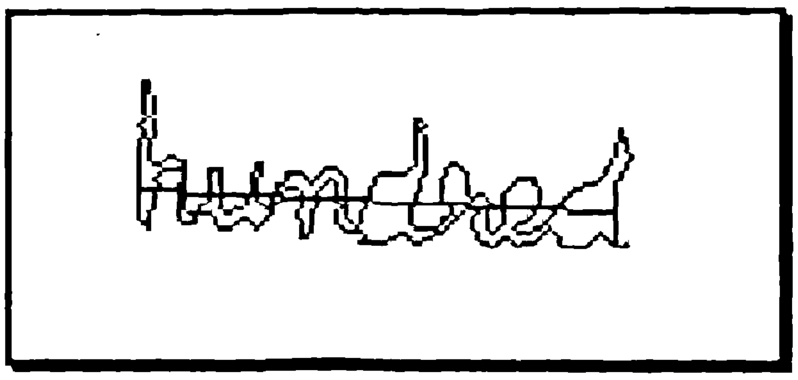
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
3	Eight	152
17	Eight	155
13	Eighteen	138
1	Eighty	158
3	Eighty	154
17	Eighty	153
19	Eighty	152

Recognized as word 'EIGHTY' after the second phase(Lower Contour Match).

Matched sample is **variant 3** of **sample eighty**.

Example 15:



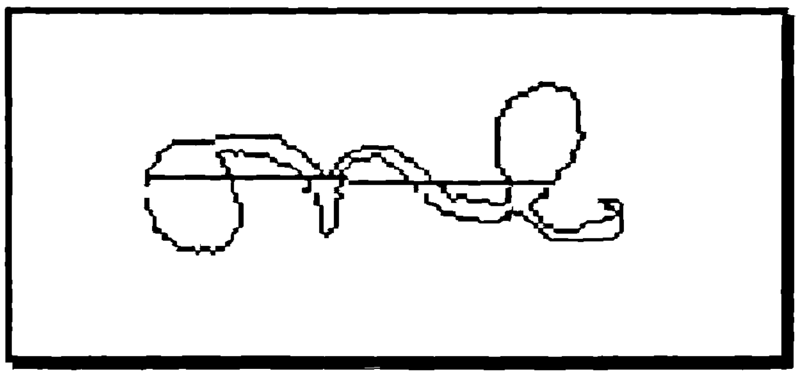
Matched list after the first phase (Upper Contour Match) :

Variant	Sample	Cost
5	Hundred	61
12	Hundred	54
18	Hundred	56
23	Hundred	63

Recognized as word '**HUNDRED**' after the second phase(Lower Contour Match). Matched sample is **variant 18 of sample hundred**.

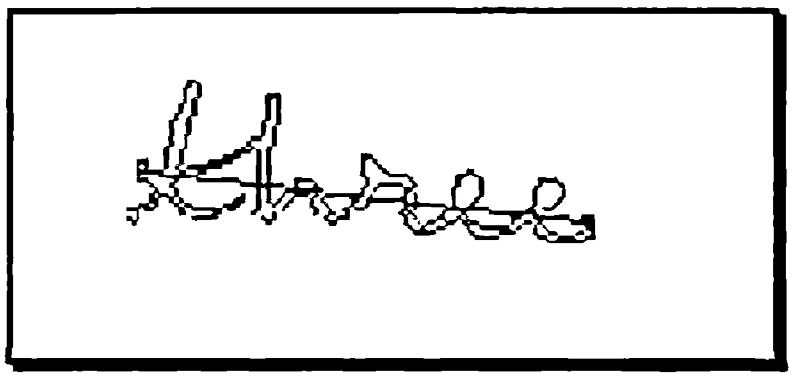
The few other samples that are correctly recognized are as follows :

Example 16 :



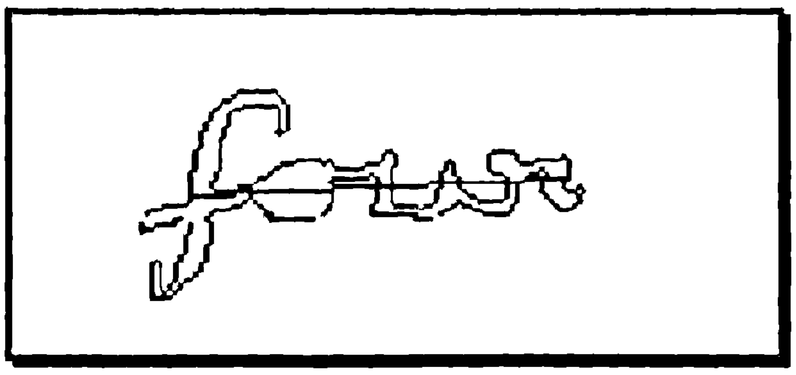
Recognized as word '**ONE**' after the second phase(Lower Contour Match). Matched sample is **variant 8 of sample one.**

Example 17:



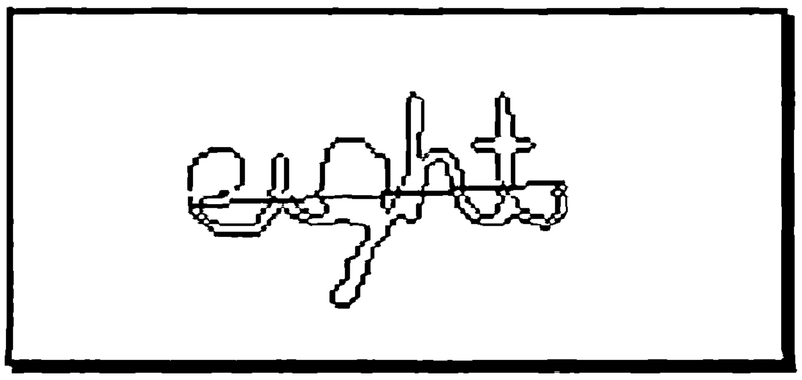
Recognized as word '**THREE**' after the second phase(Lower Contour Match). Matched sample is **variant 20 of sample three**.

Example 18.



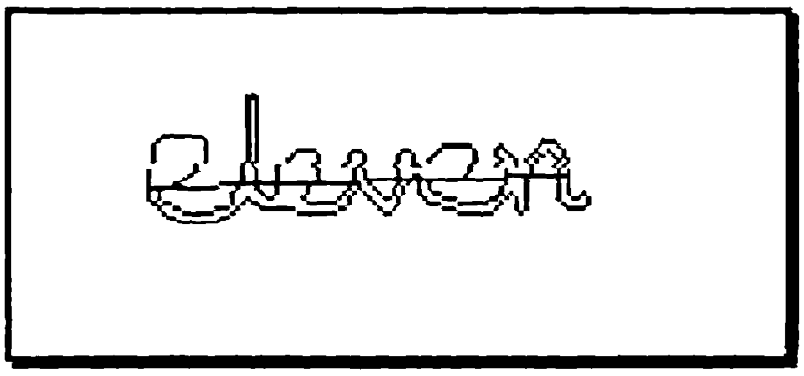
Recognized as word '**FOUR**' after the second phase(Lower Contour Match). Matched sample is **variant 12** of **sample four**.

Example 19:



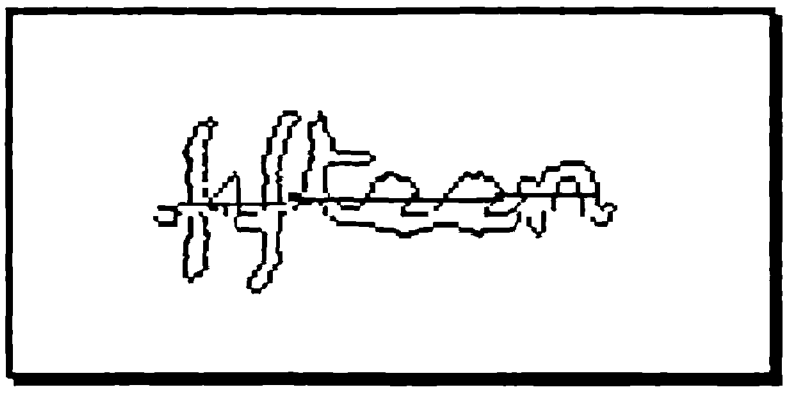
Recognized as word '**EIGHT**' after the second phase(Lower Contour Match) Matched sample is **variant 13** of **sample eight**.

Example 20:



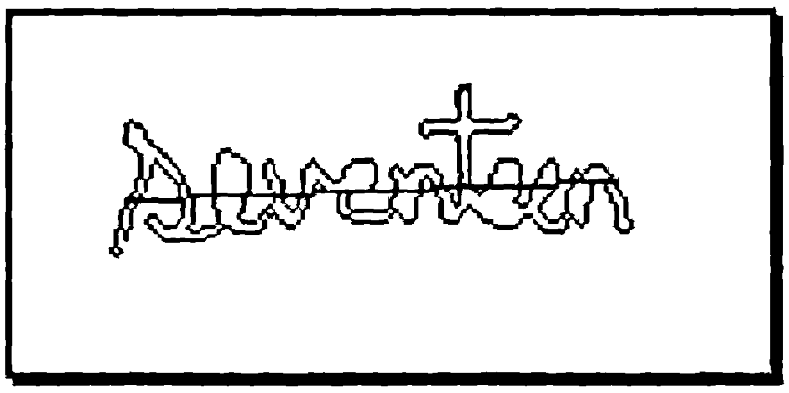
Recognized as word **'ELEVEN'** after the second phase(Lower Contour Match). Matched sample is **variant 2** of **sample eleven**.

Example 21:



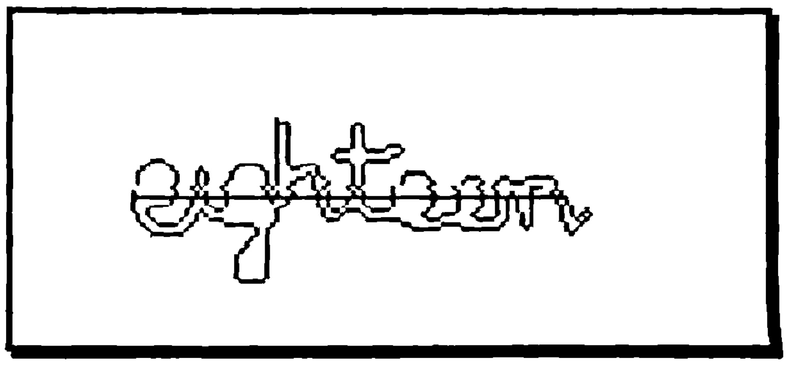
Recognized as word **'FIFTEEN'** after the second phase(Lower Contour Match). Matched sample is **variant 3** of **sample fifteen**.

Example 22:



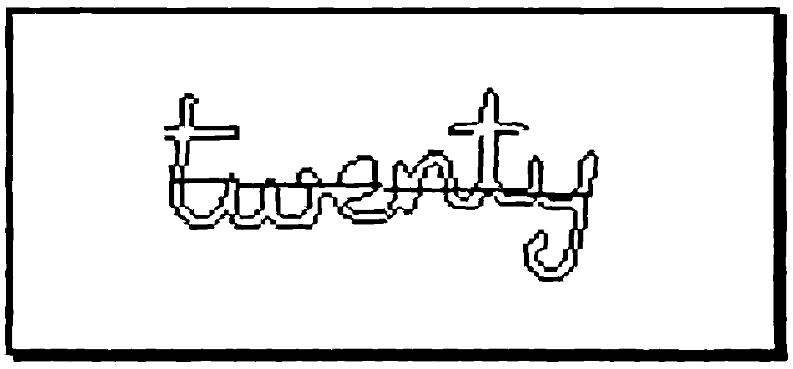
Recognized as word **'SEVENTEEN'** after the second phase(Lower Contour Match). Matched sample is **variant 3** of **sample seventeen**

Example 23:



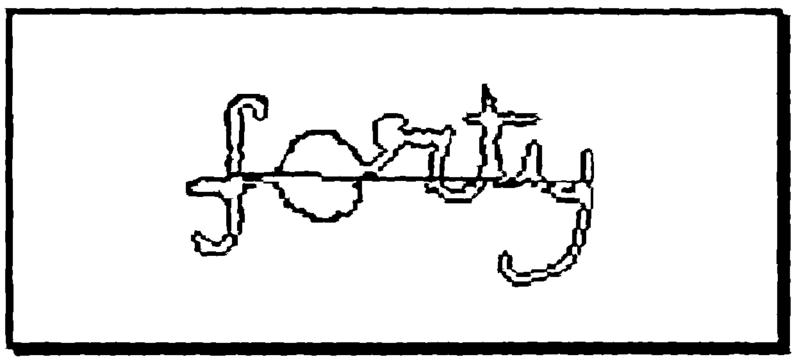
Recognized as word 'EIGHTEEN' after the second phase(Lower Contour Match). Matched sample is **variant 17** of **sample eighteen**.

Eample 24:



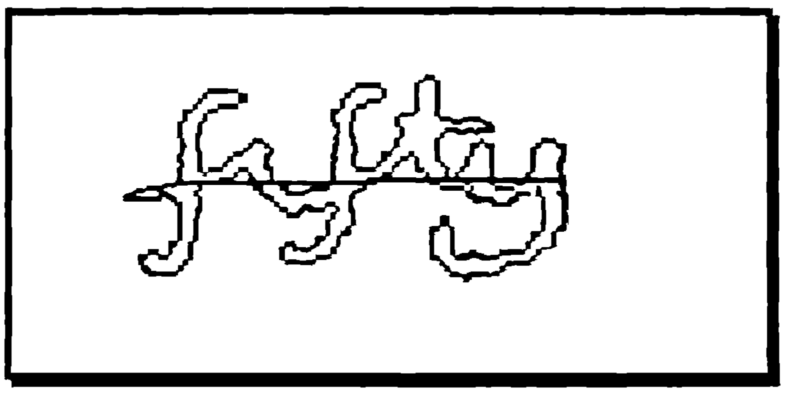
Recognized as word 'TWENTY' after the second phase(Lower Contour Match). Matched sample is **variant 16** of **sample twenty**.

Example 25:



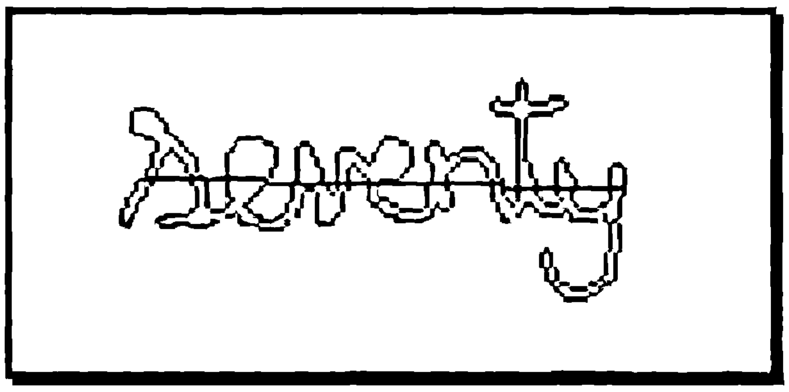
Recognized as word **'FORTY'** after the second phase(Lower Contour Match). Matched sample is **variant 12** of **sample forty**.

Example 26:



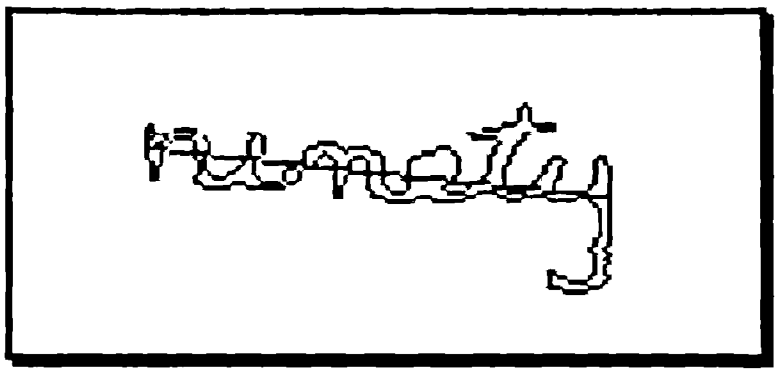
Recognized as word **'FIFTY'** after the second phase(Lower Contour Match). Matched sample is **variant 12** of **sample fifty**.

Example 27:



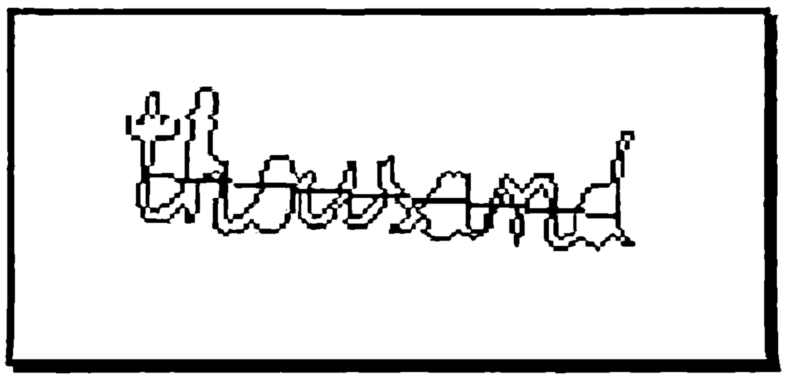
Recognized as word **'SEVENTY'** after the second phase(Lower Contour Match). Matched sample is **variant 17** of **sample seventy**.

Example 28:



Recognized as word '**NINETY**' after the second phase(Lower Contour Match). Matched sample is **variant 13** of **sample ninety**.

Example 29:



Recognized as word '**THOUSAND**' after the second phase(Lower Contour Match). Matched sample is **variant 16** of **sample thousand**.

Finally a small note on the dictionary. For each of the above examples, the minimum and the maximum costs of the input's upper contour with that of those stored in the dictionary are found out, and are given below.

Cost Estimates for the Samples in the Dictionary

INPUT SAMPLE	Minimum Cost in its Class	Maximum Cost in its Class	Maximum Cost in the Whole Dictionary
ONE	56	120	365
TWO	56	174	396
THREE	37	222	314
FOUR	48	167	349
FIVE	87	194	333
SIX	86	219	335
SEVEN	98	171	352
EIGHT	130	212	305
NINE	136	184	292
TEN	116	218	331
ELEVEN	86	174	306
TWELVE	131	212	300
THIRTEEN	52	269	331
FOURTEEN	117	194	306
FIFTEEN	116	232	338
SIXTEEN	122	229	343
SEVENTEEN	138	265	326
EIGHTEEN	98	204	288
NINETEEN	127	298	298
TWENTY	127	224	298
THIRTY	162	253	325
FORTY	116	212	305
FIFTY	102	232	321
SIXTY	155	223	321
SEVENTY	50	239	300
EIGHTY	152	235	328
NINETY	48	153	333
HUNDRED	54	189	316
THOUSAND	35	199	329

Chapter VI

Conclusions and Future directions

6.1 Conclusions

The proposed method for the recognition of handwritten cursive scripts using edit distance computation is able to

1. Partition the whole contour into two halves namely upper and lower contours, by finding a least mean square fit line.
2. Find a matched set by matching the unknown strings upper contour with all the upper contours present in the dictionary by applying edit distance technique .
3. Find a match for the lower contour of the unknown sample from within the matched sets' lower contours.
4. Finally, recognize the unknown word, in case of a tie in lower contour matching phase, by applying rotational invariant string matching technique.

The proposed algorithm is giving satisfactory results as expected Altogether some 81 unknown samples are tested of which 77 are correctly recognized. The success rate is 95%. For some of the words the proposed algorithm is not able to recognize correctly. For example 'thirteen' is recognized as 'thirty'. The reason for this is that, when the upper contours of both these samples are considered, they differ very slightly (i.e the two consecutive strokes of 'e' are misinterpreted for 'y' since we consider only the pixel level) also that the corresponding lower contours are very similar. So this very small difference in shape contributes for very less cost, thereby misinterpreting the unknown sample. The other drawback for the proposed method is that, a word that is not in the dictionary is recognized as some word in the dictionary which is similar to it. For the dictionary that is used in this work, it has not been possible to set a proper threshold value on the distance, to completely rule out a word that is not at all present in the dictionary. The other drawback for the proposed method is that, the word should have continuity. Finally, one more constraint is that, the time taken to recognize an unknown sample is slightly more.

6.2 Future directions

Some of the remedies for the improvement of the proposed method, keeping in mind the drawbacks as described in the previous section, would be :

1. Careful assignment of costs to elementary edit operations,
2. By proper smoothing of the samples (unknown as well as known samples) so that the final cost of a sample that is not in the dictionary would be high and so can be discarded,
3. Proper selection of the variants of the same word while constructing the dictionary. Select only a few variants from a set of same words that are largely varying, say six to seven. By doing so, the dictionary size would be less as well as the time taken to recognize the unknown word would be less, since the number of samples is less.
4. Sharpening of the gray level image can be useful if there are very small discontinuities present in the word.
5. Considering block substitution during matching by finding the stick length.

Bibliography

[Chow et. al.:94] **A New Dynamic Approach for Finding the contour of Bi-level Images** *Louis R. Chow, H. C. Liu, S. Y. Hsu and D.W. Wu, CVGIP : Graphical models and image processing, Vol. 56, No 6, Nov'94, pp 507-509.*

[Pavlidis:82] **Algorithms for Graphics and Image Processing** *Theodosios Pavlidis, Computer Science Press, 1982.*

[Ezzel:94] **Windows 3.1 Graphics Programming** *Ben Ezzel, PC Magazine, BPB Publications, 1994.*

[Kim & Govindaraju:96] **Efficient chain code based image manipulation for handwritten word recognition** *Gyeonghwan Kim and Venu Govindaraju, Recognition, in Proc. of the SPIE symposium on electronic imaging science and technology (Document Recognition III), San Jose, CA, vol. 2660, pp. 262--272, February 1996.*

[Bunke & Bühler:93] **Applications of approximate String Matching to 2D shape recognition** *H.Bunke and U.Bühler, Pattern Recognition, Vol.26, No.12, pp. 1797-1812, 1993.*

[Oommen & Kashyap:96] **A Formal Theory For Optimal and Information Theoretic Syntactic Pattern Recognition** *B.J.Oommen and R.L.Kashyap, International symposium on Syntactic and Structural Pattern Recognition, Leizig, Germany August 96.*

[Lin et al.:92] **Contour Shape Description Based On an Arch Height Function** *Yuia Lin, Jiqing Dou, Hongmei Wang, Pattern Recognition, Vol. 25, No 1, pp 17-23, 1992.*

[Koplowitz & Plante:95] **Corner Detection For chain coded curves** *Jack Koplowitz and Stephen Plante, Pattern Recognition, Vol. 28, No. 6, pp. 843-852, 1995.*

[Lecolinet & Baret 93] **Cursive Word Recognition : Methods And Strategies** *Lecolinet E. and Baret O., Fundamentals in Hand Writing Recognition, NATO Advanced Institute, Chatean de Bonas, France, June 1993.*

[Duda & Hart:73] **Pattern Classification and Scene Analysis** *R.O.Duda and P.E.Hart, Wiley & Sons, 1973.*

[Wagner & Fischer:74] **The string-to-string correction problem** *R.A.Wagner and M.J.Fischer, Journal of ACM 21, No. 1, pp.168-173, 1974.*

[Riseman & Hanson:74] **A Contextual Post Processing System For Error Correction Using Binary n-grams** *Riseman E.M. and Hanson A.R., IEEE Transactions on Computers, Vol. C-23, pp. 480-493, May 1974.*

[Sinha & Prasada:88] **Visual Text Recognition Through Contextual Processing** *Sinha R,M.K. and Prasada B., Pattern Recognition, Vol. 21, No. 5, pp. 463-469, 1988.*

[Madvanath:96] **The Holistic Paradigm in Handwritten Word Recognition and Its Application to Large and Dynamic Lexicon Scenarios** *Sriganesh Madvanath, http://www.cedar.buffalo.edu/Publications/Diss_Abs/abstracts.html, Ph.D Dissertation Abstracts From CEDAR,*

[Bozinovic & Srihari:89] **Offline Cursive Script word Recognition** *Srihari S.N. and Bozinovic R.M., IEEE Transactions on PAMI, Vol. 11, No. 1, pp 68-82, Jan 1989*

[Maes:91] **Polygonal Shape Recognition Using String Matching Techniques** *Maes M., Pattern Recognition, Vol. 24, pp. 433-440, 1991.*

[Dinstein et. al.:91] **Parallel Algorithms for contour based 2D Shape Recognition** *I Dinstein, G.M.Landu and G.Guy, Pattern Recognition, Vol 24, pp.929-942 , 1991.*

[Srihari & Rohini:96] **Overview** *Sargur N. Srihari & Rohini K. Srihari, State University of New York at Buffalo, New York,USA, CEDAR http://www.cse.ogi.edu/CSLU/HLTsurvey/ch2node3.html, April '96.*

[Belaid 96] **OCR Print** *Abdel Belaïd, CRIN CNRS & INRIA, Nancy, Lorraine, France http:// www.cse.ogi.edu/CSLU/HLTsurvey ch2node5.html, April '96.*

[Faure & lecolinet:96]**OCR: Handwriting** *Claudie Faure & Eric Lecolinet, Télécom Paris,France, http://www.cse.ogi.edu/CSLU/ HLTsurvey ch2node6.html,April '96.*

[Phillips:94] **Image processing in C** *Dwayne Phillips,R&D Books 1994.*

[Srihar & Lam:95] **Character Recognition** *Sargur N. Srihari & Stephen W. Lam Center of Excellence for Document Analysis and Recognition State University of New York, USA, CEDAR http: // www.cedar. buffalo.edu/Publications/TechReps/OCR/introcrfg.html Sept 95.*

Appendix A

TIFF File Format

Here this section of the Appendix give a brief description about the TIFF (Tagged Image File Format) file format.

There are many standard image file formats into which a image can be stored (different standard file formats are discussed in chapter two). One of which that is chosen for the present work is a TIFF file format, because most of the scanner manufacturers support this standard. Also that when compared to other standards, retrieving information from uncompressed 256 color gray level TIFF image is easy to implement. A scanner is used to scan a 256 color gray level image. For our problem only bi-level image is required. To have a binary image from a gray level image, the information present in a gray level TIFF file has to be read first and then by using proper thresholding has to convert it into bi-level image. That needs some knowledge regarding the TIFF specifications : Baseline TIFF and TIFF extensions[Phillips:94]. All TIFF files should be totally compatible with the baseline level. The extensions go beyond the baseline in several ways, providing advanced capabilities. A TIFF file has three components , Image File Header, Image File Directory and Image Data itself. Image file header .

<table>
<tr><th>Base Number</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td rowspan="2"><b>Header
Contents</b></td><td>49</td><td>49</td><td>2A</td><td>00</td><td>08</td><td>00</td><td>00</td><td>00</td></tr>
<tr><td colspan="2"><b>Byte Order</b></td><td colspan="2"><b>TIFF identifier</b></td><td colspan="4"><b>Offset of the first IFD</b></td></tr>
</table>

TIFF header from a file created on an IBM Compatible PC

Base Number	0	1	2	3	4	5	6	7
Header Contents	4D	4D	00	2A	00	00	00	08
	Byte Order		TIFF identifier		Offset of the first IFD			

TIFF header from a file created on an Apple Macintosh

Image File Directory (IFD)

The IFD has the following structure :

Byte 0-1 Tag identifying the field

Byte 2-3 Field type

Byte 4-7 Number of values, or count of the type

Byte 8-11 Offset into the file where the values begin.

Required fields for Gray scale Images are:

Tag Name	Decimal(Hex)	Type	Value
Image Width	256(100)	Short/Long	
Image Length	257(101)	Short/Long	
Bits Per Sample	258(102)	Short	4/8
Compression	259(103)	Short	1/32,773
Strip Offsets	273(111)	Short/Long	
Rows Per Strip	278(116)	Short/Long	
Strip Byte Counts	279(117)	Short/Long	
X Resolution	282(11A)	Rational	
Y Resolution	283(11B)	Rational	
Resolution Unit	296	128	1/2/3

A Bi-level image has all the above fields except Bits Per Sample field

Appendix B

Data Set

Here in this section of the Appendix, all the sample images that were taken as Data set, and also the procedure that has to be followed, to construct the dictionary from this Data set are presented. All together there are 24 image files written by 24 different people. These image files are combined into 8 sets of 3 images each. They are :

One two three four five six seven eight
nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen ninteen
twenty thirty forty fifty sixty seventy
eighty ninety hundred thousand

one two three four five six seven eight
nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen
twenty thirty forty fifty sixty seventy
eighty ninty hundred thousand

one two three four five six seven eight
nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen
twenty thirty fourty fifty sixty seventy
eighty ninety hundred thousand

one two three four five six seven eight

nine ten eleven twelve thirteen fourteen

fifteen sixteen seventeen eighteen nineteen

twenty thirty forty fifty sixty seventy

eighty ninety hundred thousand

one two three. four five. six seven eight

nine ten eleven twelve thirteen fourteen fifteen

sixteen seventeen eighteen nineteen twenty thirty

forty fifty sixty seventy eighty ninety

hundred thousand.

one two three four five. six seven eight nine ten

eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen

twenty thirty forty fifty sixty seventy eighty ninety hundred

thousand.

one two three four five six seven
eight nine ten eleven twelve
thirteen fourteen fifteen thousand
sixteen seventeen eighteen nineteen
twenty thirty forty fifty sixty
Seventy eighty ninety hundred

one two three four five six seven
eight nine ten eleven twelve
thirteen fourteen fifteen sixteen
seventeen Eighteen nineteen twenty
thirty forty fifty sixty seventy
Eighty ninety hundred thousand

One two three four five six
seven eight nine ten eleven
twelve thirteen fourteen fifteen sixteen
seventeen eighteen nineteen twenty
thirty forty fifty sixty
seventy eighty ninety hundred
thousand

One two three four five six
seven eight nine ten eleven
twelve thirteen fourteen fifteen
sixteen seventeen eighteen nineteen.
twenty thirty forty fifty sixty
seventy eighty ninety hundred
thousand

One- two three four five Six
Seven eight nine ten eleven
twelve thirteen fourteen fifteen
sixteen seventeen eighteen ninteen
twenty thirty forty fifty sixty
seventy eighty ninety
thousand hundred

one two three four five six seven
eight nine ten eleven twelve thirteen
fourteen fifteen sixteen seventeen eighteen
nineteen twenty thirty forty fifty
sixty seventy eighty ninety hundred
thousand

one two three three four five six seven eight
nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen
twenty thirty . forty fifty sixty
seventy eighty ninety hundred thousand

One two three four five six seven
eight nine ten eleven twelve thirteen
fourteen fifteen sixteen seventeen eighteen
nineteen twenty thirty forty fifty
sixty seventy eighty ninety hundred
thousand

one two three four five six seven
eight nine ten eleven twelve
thirteen fourteen fifteen sixteen
seventeen eighteen nineteen twenty
thirty ~~foorty~~ forty fifty
sixty seventy ninety eighty
hundred thousand.

one two three four five six
Seven eight nine ten eleven
twelve thirteen fourteen fifteen Sixteen
Seventeen eighteen nineteen twenty
thirty forty fifty Sixty Seventy eighty
ninety hundred thousand

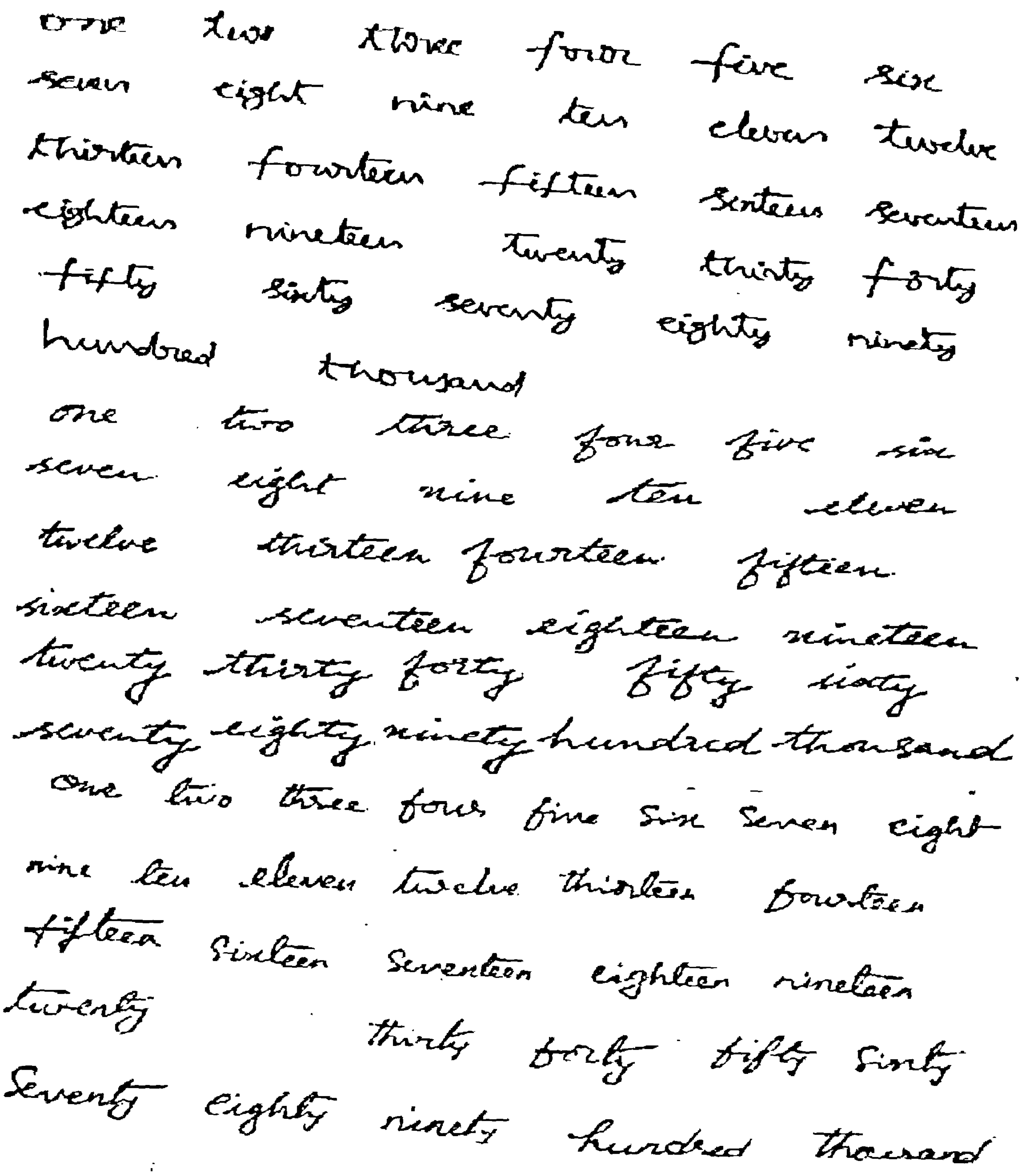
one two three four five six
seven eight nine ten eleven
twelve thirteen fourteen fifteen
sixteen seventeen eighteen
nineteen twenty thirty forty
fifty sixty seventy eighty
ninety hundred thousand

One two three four five six
seven eight nine ten eleven twelve
thirteen fourteen fifteen sixteen
seventeen eighteen nineteen twenty
thirty forty fifty sixty seventy eighty
ninety hundred thousand

one two three bour bive six
seven eight nine ten eleven
twelve thirteen fourteen bibteen sixteen
seventeen eighteen nineteen twenty
thirty borty bibty sixty seventy eighty
ninety hundred thousand

One Two three four five six seven
eight nine ten eleven twelve thirteen
fourteen fifteen sixteen seventeen
eighteen nineteen twenty Thirty
forty fifty sixty seventy
eighty ninety hundred thousand

one two three four five six
seven eight nine ten eleven
twelve thirteen fourteen fifteen sixteen
seventeen eighteen nineteen twenty thirty
forty fifty sixty seventy eighty ninety
hundred thousand



one two three four five six
seven eight nine ten eleven twelve
thirteen fourteen fifteen sixteen seventeen
eighteen nineteen twenty thirty forty
fifty sixty seventy eighty ninety
hundred thousand

one two three four five six
seven eight nine ten eleven
twelve thirteen fourteen fifteen
sixteen seventeen eighteen nineteen
twenty thirty forty fifty sixty
seventy eighty ninety hundred thousand

One two three four five Six Seven eight
nine ten eleven twelve thirteen fourteen
fifteen Sixteen Seventeen eighteen nineteen
twenty thirty forty fifty Sixty
Seventy eighty ninety hundred thousand

Once all the image files are present (i.e., Data Set), the next step is to ;onstruct a dictionary from these image files. The first step in constructing the lictionary is to cut all the images into smaller ones (Templates) such that each