1-1-2006

Hierarchical approach for character recognition

Koushik Reddy Damera
University of Nevada, Las Vegas

HIERARCHICAL APPROACH FOR CHARACTER RECOGNITION

By

Koushik Reddy Damera

Bachelor of Engineering in Computer Science and Engineering
Osmania University, Hyderabad, India
June 2004

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2006

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

NOVEMBER 13 , 20 06

The Thesis prepared by

KOUSHIK REDDY DAMERA

Entitled

HIERARCHICAL APPROACH FOR CHARACTER RECOGNITION.

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

1017-53

ABSTRACT

Hierarchical Approach for Character Recognition

By

Koushik Reddy Damera

Dr. Evangelos Yfantis, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

This research mainly focuses on recognizing the handwritten character. Several efficient algorithms have been developed by us so far to separate: the handwritten characters from printed text character, the lines, the words, and each character. In this thesis, we concentrate on how to increase the efficiency of recognition of segmented handwritten characters. Certain characters share common features unique to each other, different from the rest of characters. These subsets of characters with common features are then further analyzed by the classifier thereby reducing the number of comparisons that are required. This results increasing the speed of character recognition.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

ACKNOWLEDGEMENTS

First of all I would like to express my gratitude to Professor Evangelos A. Yfantis for supervising me in my thesis and providing the invaluable support and guidance without which this thesis would not have been complete.

I would like to thank Dr. Ajoy K Datta, Dr.Yahoowan Kim and Dr. Venkatesan Muthukumar for accepting to be my committee members in short notice. Finally I would like to thank all my friends who were there to help and support me during the development of this thesis work.

CHAPTER 1

INTRODUCTION

Handwritten character recognition is a process designed to translate images of handwritten or typewritten text into machine-editable text, or to translate pictures of characters into a standard encoding scheme representing them .There are two types of recognition one is offline character recognition in which recognition is done using the features extracted from the scanned image where as in the online character recognition it is done by following the movements of the pen online. The technology is successfully used by businesses which process lots of handwritten documents, like insurance companies, postal service etc. But the recognition of the handwritten characters is complicated due to the following reasons

1) From user to user the same character may vary in size, shape and style. Even the same user may write in different patterns from time to time.
2) Like any other images, the character that are written may subject to spoilage due to noise.

Hence handwritten characters are first preprocessed to increase the accuracy and efficiency of the recognizers. The following algorithms are implemented in preprocessing

1) Binary image conversion
2) Speckle Removal and Noise Removal

Once the preprocessing is done then comes the feature extraction of the characters which are used by the classifiers. There are many feature extraction algorithms but the important key is the selection of the efficient feature that would help in implementation of the character recognizer. Once the features are extracted from all the characters, they are given to the classifier for recognition.

1.1 Problem Definition

Handwritten characters are difficult to recognize since the each user writes the same character in different ways with different sizes and different shapes. Before the recognition is done it has to be separated from the printed text and the background. Once the character is separated then we can recognize. Even though lot of research is going on in this field, it is very difficult to attain high accuracy. Hence there are lot of classifiers that are been implemented to increase the accuracy. But the time consumption for these processes is more since each character features are to be compared with others. In order to increase the efficiency we came up with the following novel concept.

Even though the different user writes the character in different sizes and the different shapes but the each character will be having the some features which are present in what ever shape and size user writes. The basic idea of our thesis is the characters that are having the same features will be grouped together and the characters that not having the same features are eliminated so that the recognizer uses those characters which having the same features for the recognition process avoiding the other characters which are not a like .For example when you take character 'A' it will be having the two tips at the bottom. And if you take the character 'I 'it will be having only one tip at the bottom.

Hence you can climate the 'I' characters when you are looking for 'A'. Similarly for some other character which can be found out by using our algorithm.

1.2 Objective

The main objective of our work is to develop the optimized preprocessing methods and to develop an algorithm which would help in classifying the characters which are having the common features and which don't have the common features when you are looking for a specific character. There by reducing the number of comparisons that are required and increase the speed of the recognition.

CHAPTER 2

LITERATURE REVIEW

Optical character recognition abbreviated as OCR is a way in which the handwritten text images, printed text images into machine editable format. In 1929, G. Tauschek obtained a patent on OCR in Germany, followed by Handel who obtained a patent on OCR in USA in 1933 (U.S. Patent 1,915,993). Tauschek was in 1935 also granted a US patent on his method (U.S. Patent 2,026,329).These are the first concepts in the field of the OCR. The principle used at that time is the template mask matching. This used the optical and mechanical template matching. Light is been passed through the mechanical masks is captured by a photo detector and is scanned mechanically when exact match occurs light fails to reach the detector and so machine recognizes the character printed on paper. Since there is vast variety of the handwritten characters the template matching for the handwritten characters is very difficult .Therefore the template matching is been followed by the structural analysis of the hand written characters. Structural analysis of the character is a way in which the geometric orientation of the characters is found out and the features of the characters are extracted. There are different methods proposed and published for the efficient feature extraction method. Since every year different methods of extraction of features are been adding up to the research in the field of OCR it is very difficult to tell which method is efficient. And the efficiency of the OCR not only

depends on feature extraction method but also classifiers. Each feature extraction method requires different type of classifiers. And we cannot compare the results because the recognition process is done on different data sets .Once the features are extracted from the characters, the key decision is the selection of the efficient features that are to be used in the recognition process. Once the efficient features are been extracted, then these features are been used in by the classifier for the recognition of the characters. Different classification methods are been used in recognition process.

These are the following classification methods.

2.1) Statistical methods

2.2) Artificial Neural Networks

2.3) Kernel Methods

2.4) Multiple Classifier Combination.

2.1 Statistical Methods

Statistical methods are based on the Bayes Decision Theory. There are different decision theories. Bayes decision theory is optimal and is base on the popular Bayes rule

$P(x/y) = p(y/x)p(x)/p(y)$

Assuming the Gaussian density the Baysian Discriminant method is reduced to one of the following methods

2.1.1) Linear Discriminant Method [3]

2.1.2) Quadratic Discriminant Method

2.1.3) Modified Quadratic Discriminant Method[15]

2.2 Artificial Neural Networks

Artificial Neural Networks is been used in successfully in pattern recognition[17][18].They are many artificial neural networks that are been used. They are

2.2.1 Feed Forward Neural Network

2.2.2 Radial Basis Function

2.2.1 Feed Forward Neural Network

Feed forward networks have the following characteristics:

1) Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.
2) Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next, and this explains why these networks are called feed-forward networks.
3) There is no connection among perceptrons in the same layer

2.2.2 Radial Basis Function

After the FF networks, the Radial Basis Function (RBF) network comprises one of the most used network models. Figure 2.1 illustrates an RBF network with inputs X_1,..., X_n and output $\hat{y}$. The arrows in the figure symbolize parameters in the network. The RBF network consists of one hidden layer of basis functions, or neurons. At the input of each neuron, the distance between the neuron center and the input vector is calculated. The output of the neuron is then formed by applying the basis function to this distance.

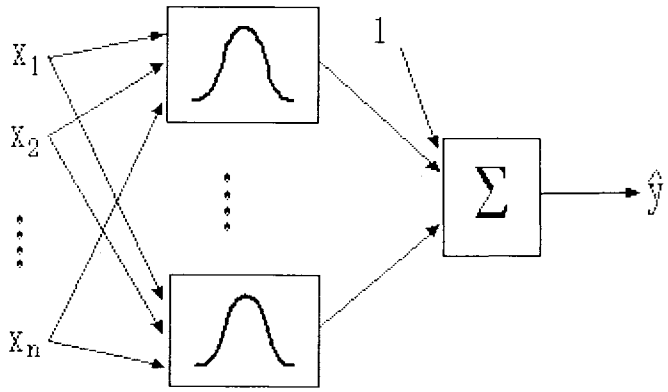


Figure 2.1 An RBF network with one output.

2.3 Kernel Methods (Support Vector Machines)

Support vector machines [11, 12] is the significant kernel method that is been followed. SVM is a binary classification with discriminant function being weighted combination of kernel functions over all training samples. SVM are used for classification and regression. Due to the high complexity in implementing the SVM are used for small data classification.

2.4 Multiple Classifier Combination

In order to increase the accuracy and efficiency of the optical character recognizer the single classifier is been replaced by the couple of classifiers[14] . If the multiple classifiers are parallel then the accuracy will be improved. If the multiple classifiers are sequential then the speed of the classifier improves.

In the all above methods for the recognition of the characters the algorithm has to be trained with huge database of the characters and the features of the each character input has to be saved .Then the features that are been extracted are to be given to the classifier .

In the classification process the classifier that is been used takes the features and compares with the other character features and then the recognition is done.

Flow chart that is been followed

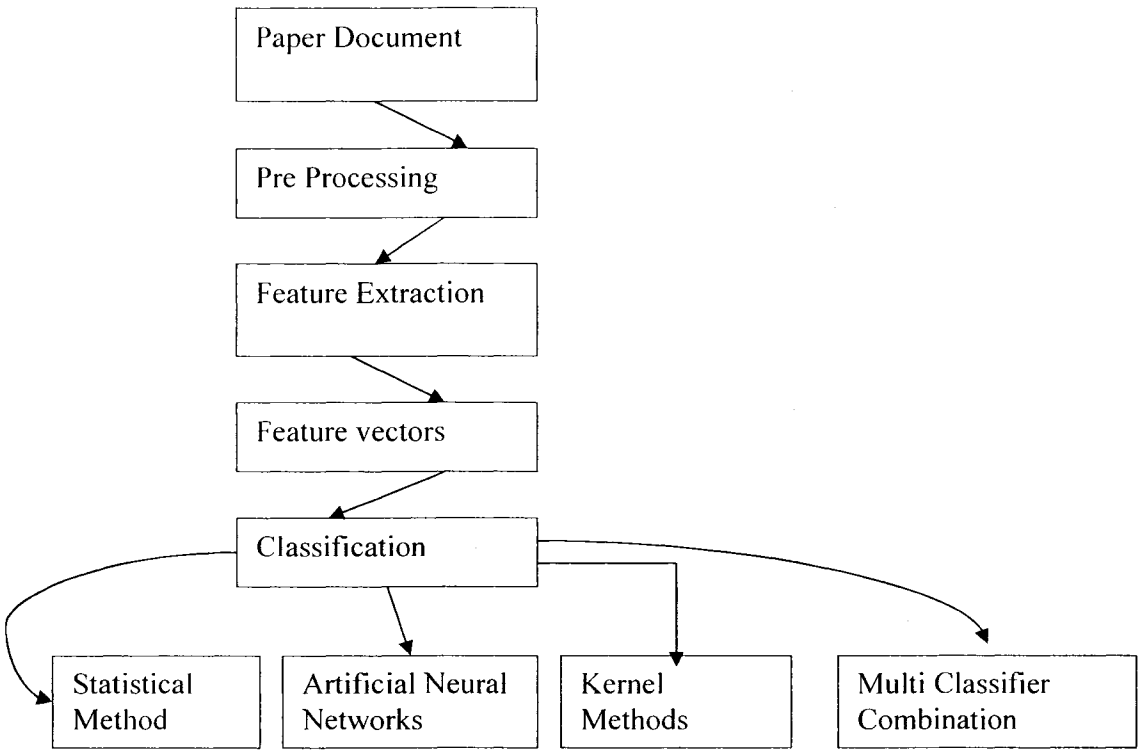


Figure 2.2 Flowchart General Approach

Hence we came up with the new idea to reduce the time for the recognition process. This can be done by reducing the number of alphabets that has to be compared with the

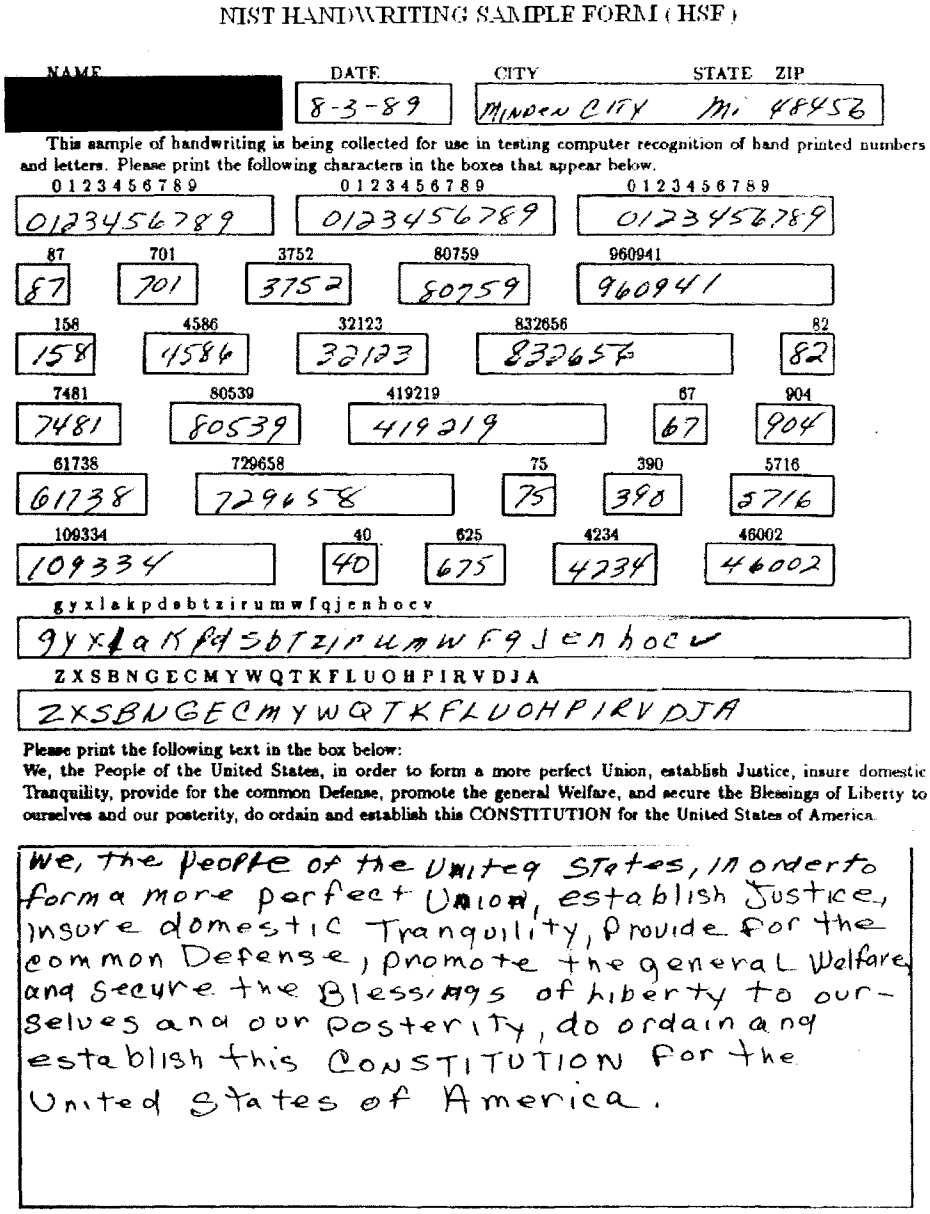
other alphabets. Since some of the features of the characters are completely different from the features of the other alphabets.

2.5 Input Database

The efficiency and accuracy of the recognition algorithms that are been developed can be tested against standard database that have the different forms of characters. Since each user writes the character in different shapes and styles, we require the test database that contains the characters that are written in most shapes, styles and sizes. Hence we took the National Institute of Standards and Technology (NIST) database as the input for testing the functioning and the accuracy of the algorithms that are been used.

National Institute of Standards and Technology

National Institute of Standards and Technology provides the databases for the character recognition, finger print recognition, face recognition. The database that is been used is "Special Database 19-NIST Hand printed Forms and Characters Database", shortly known as SD19 database [7]. It contains the Handwritten Sample Forms(HSF) and isolated characters that are been stored by the class that they belong to. Isolated characters are divided into upper case letters, lower case letters, numerical numbers. Hence you can combine all of them to form the total database.


NIST HANDWRITING SAMPLE FORM (HSF)

NAME [redacted] DATE 8-3-89 CITY Minden City STATE MI ZIP 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9

87 | 701 | 3752 | 80759 | 960941

158 | 4586 | 32123 | 832656 | 82

7481 | 80539 | 419219 | 67 | 904

61738 | 729658 | 75 | 390 | 5716

109334 | 40 | 625 | 4234 | 46002

g y x l a k p d o b t z i r u m w f q j e n h o c v

Z X S B N G E C M Y W Q T K F L U O H P I R V D J A

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of liberty to ourselves and our Posterity, do ordain and establish this CONSTITUTION for the United States of America.

Figure 2.3 Sample HSF form.

These isolated characters are of size 128 * 128 pixels. The Special Database 19 (SD 19) contains the 3699 Handwritten Sample Forms and 814255 segmented hand printed digit and alphabetic characters form the HSF forms .The isolated characters are divided in to 62 classes corresponding to "A" to "Z", "a" t o "z" and "0" to "9".

2.6 Pre-processing

Before the recognition is done the preprocessing of the characters is to be performed to increase the accuracy of the recognition. Following are the preprocessing algorithm

2.6.1 Binary image conversation.

2.6.2 Speckle removal and Noise removal

2.6.1 Binary Image Conversation

Once the character is extracted, the extracted character images are converted into binary images. Binary image is an image in which the pixel intensity is either 1(white) or 0(Black).Here is the procedure for converting the image to binary image. The image is first converted into gray scale image using the formula

$$[Y] = 0.299*R+0.587*G + 0.114*B$$

Once the image is converted into gray scale we will find whether the intensity of the each pixel whether it is greater than the threshold value or not. If the intensity is greater than threshold value then the pixel value is stored as black pixel else the pixel is stored as the white pixel. Hence the character is converted into black with white back ground.

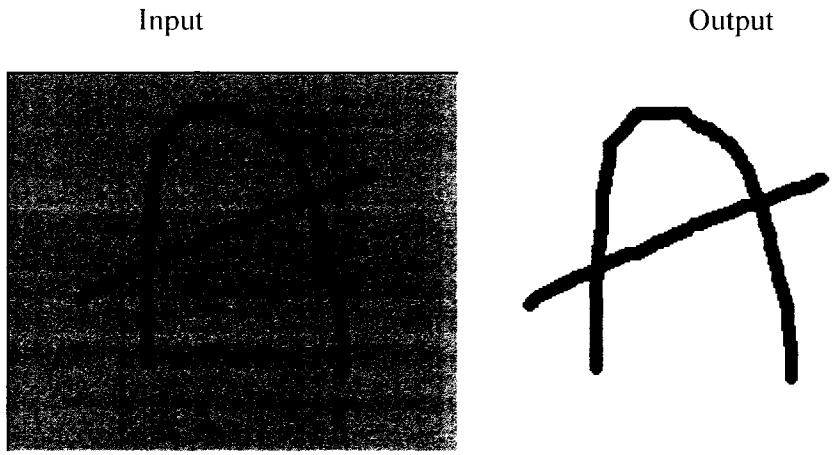
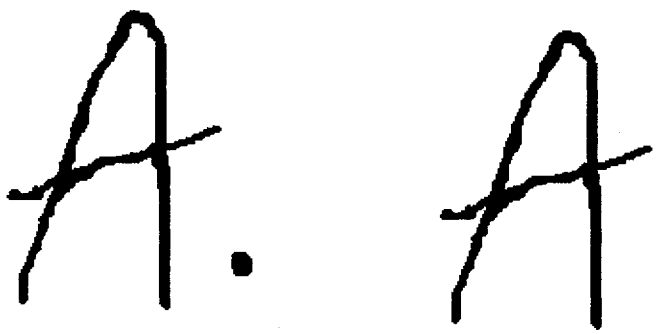


Figure 2.4 Binarization

2.6.2 Speckle Removal and Noise Removal

Spurs and dots that are present in the image should be removed to increase the accuracy of the algorithm. Size filter is been used to remove the spurs and dots present in the image. First it finds the number of components present in the image, and then it calculates the areas of those components. Once all the areas of all components is found out then the mean of those areas and standard deviation is calculated. Now the value of those components whose area is less than (mean – standard deviation/ 2) is considered to be the speckle in the image and are removed from the image.



Input Output

Figure 2.5 Speckle Removal

Once the spurs and dots are remove form the image, 3 by 3 Gaussian filter mask is been applied to smoothen the image.

CHAPTER 3

THINNING

Thinning is process in which object image is been thinned to one pixel .The final output of the image will be the image with pixel width one. Thinning is applied to the binary image. It is very important in image process to extract the features of the image.

3.1 Hilditch Algorithm

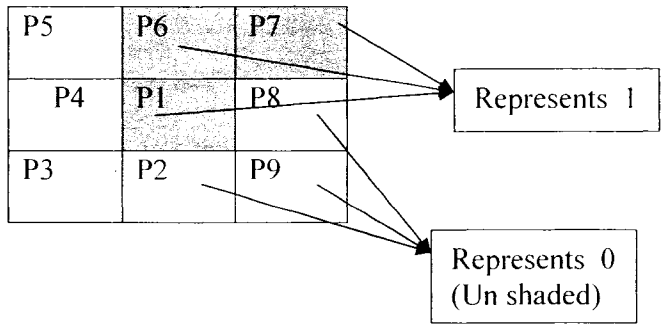
The algorithm that is followed is Hilditch thinning algorithm. We consider the 3 by 3 pixels at a time. Hence 8 neighborhood of each pixel is taken to decide whether that pixel has to remain in the final out put or it has to be peeled off to thin the character. Hence we arranged the pixels in the clock wise direction as shown below.

P5	P6	P7
P4	P1	P8
P3	P2	P9

Functions that are used

1) NZ(P1) = number of non-zero neighbors of P1

2) NT(P1) = number of 0,1 patterns in the sequence p2,p3,p4,p5,p6,p7,p8,p9,p2



The algorithm that is been used here is 3 by 3 widow version. Hilditch algorithm repeats it process when ever the black pixel is converted to white pixel. The same process is repeated on the new pixel values until the there is no change in the pixels. Hence Hilditch algorithm is parallel sequential algorithm. It is parallel because it checks all the pixels at one pass and the decision is made whether to keep the pixel or not. It is sequential because the process is repeated until no changes are done to pixels. Following are conditions that are used to decide whether pixel is to be remained in the final thinned output or not.

Conditions

1) 2 < = NZ(p1)

2) NT(p1)=1

3) p2.p4.p8=0

4) p2.p4.p6=0

When the above conditions are satisfied the pixel is converted from black to white (i.e. the pixel is been peeled off). The same procedure is been followed until there is no peeling off of the pixels.

Condition 1: 2<=NZ (P1)

This condition finds out the number of non zero pixels that are present in the 8 neighbor hood of the pixel P1. If the value is greater than 2 it ensures that no end-point pixel and no isolated one be deleted.

P5	P6	P7
P4	P1	P8
P3	P2	P9

P5	P6	P7
P4	P1	P8
P3	P2	P9

As the picture makes it clear, if NZ (P1)=1, then P1 is a skeleton tip-point and should not be deleted. If NZ (P1) =0, then P1 is an isolated point and should also be kept .

Condition 1: NT (P1) =1

This condition is been used to test the connectivity .If the value of NT(P1)>1 and the pixel is been removed then the connectivity is lost. But the result of this algorithm is to

maintain the connectivity. In the following picture the value of NT (P1) is greater than 1.Hence the removal of the pixel will make the image disconnected.

NT(P1)=2

P5	P6	P7
P4	P1	P8
P3	P2	P9

NT(P1)=2

P5	P6	P7
P4	P1	P8
P3	P2	P9

Condition 3 : *p2.p4.p8 = 0*

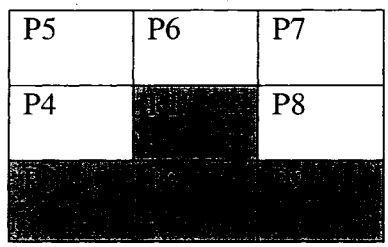
P2*P4*P8=0.

P5	P6	P7
P4	P1	P8
P3	P2	P9

Condition 4 : *p2.p4.p6 = 0*

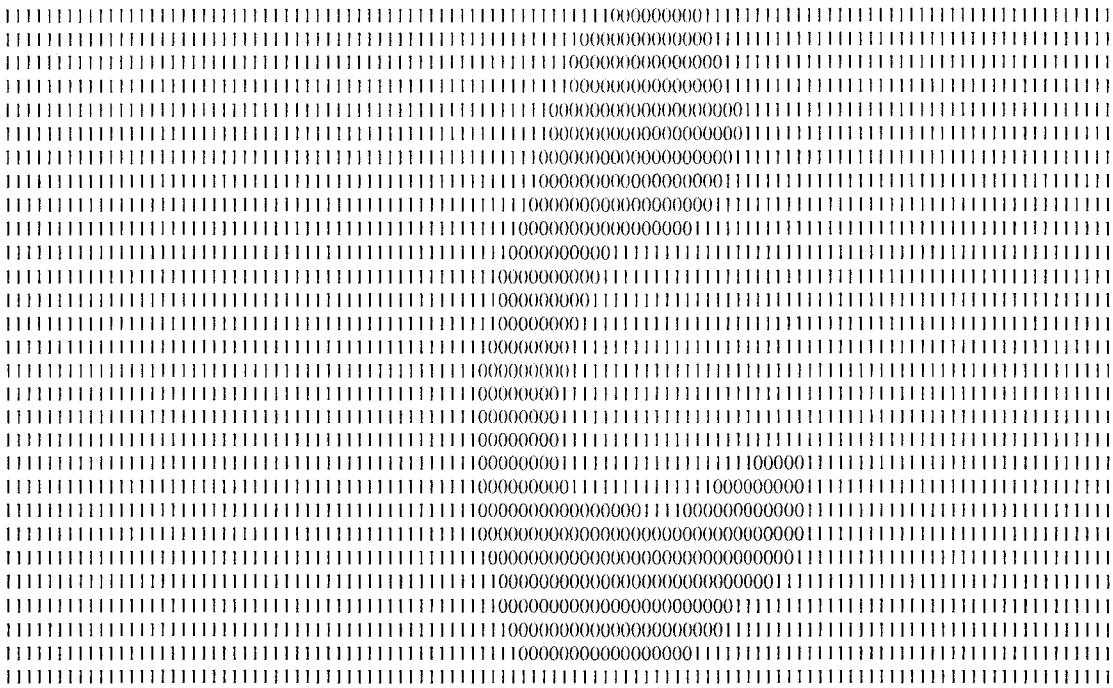
P2*P4*P6=0

P5	P6	P7
P4	P1	P8
P3	P2	P9



In the case above NZ (P1)>=2 and NT (P1) = 1 and P2*P4*P6=0 and P2*P4*P8=0.Since all the conditions are satisfied the pixel P1 is eroded from the image. This process is continued for all the pixels. Every time the pixel is been removed from the image the new pixels values are taken as an input and the same process is continued till no values of the pixels are been changed. This process will result in the skeletonization of the character. Since we have the character written in black color with white back ground ,the above condition works only if the character written in white (1) with black (0) back ground .Hence before implementing the above algorithm the image color should be changed, i.e. black pixel has to be converted to white and white pixel has to be converted to black. Once the thinning is done then we can restore the color , i.e. the black color pixel has to be stored as white and white color has to be stored as black

Input



Output

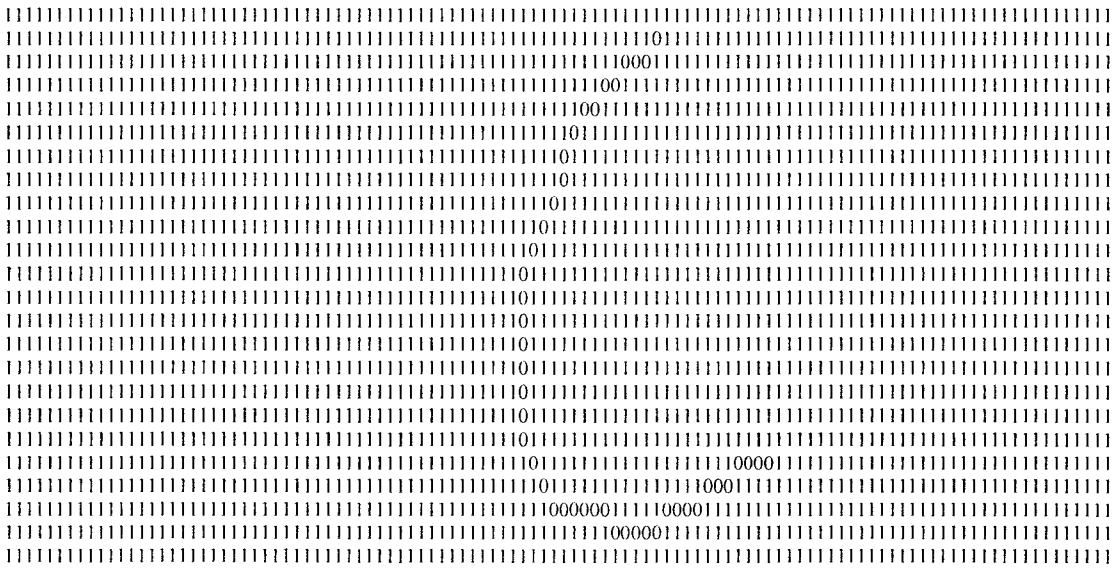


Figure 3.1 Thinning

CHAPTER 4

FEATURE EXTRACTION

4.1 Feature Extraction

Different feature extraction methods have been published for the extraction of the features of the characters. But the efficient features should be used for the classification of the characters. Before the extraction of the features the image may be divided into parts horizontally and vertically to extract the features locally. This done because the some features are same for different characters when the features are extracted for a character as a whole. For example if you take character 'A' and character 'V' the total number of tips in this case is 2 for both the characters. But if you take the number of tips for 'A' both the tips are located in the bottom part of the character and for 'V' they are located in the top part of the character. Hence the character is divided into blocks.

Features that are extracted are

4.1.1 Extreme points

4.1.2 Number of Tips

4.1.3 Change in color of horizontal pixels at midpoint vertically

4.1.4 Change in color of vertical pixels at midpoint horizontally

4.1.1 Extreme Points

Extreme Points is used to find out the size of the image and where is it located so that you can split the character into vertical and horizontal parts.

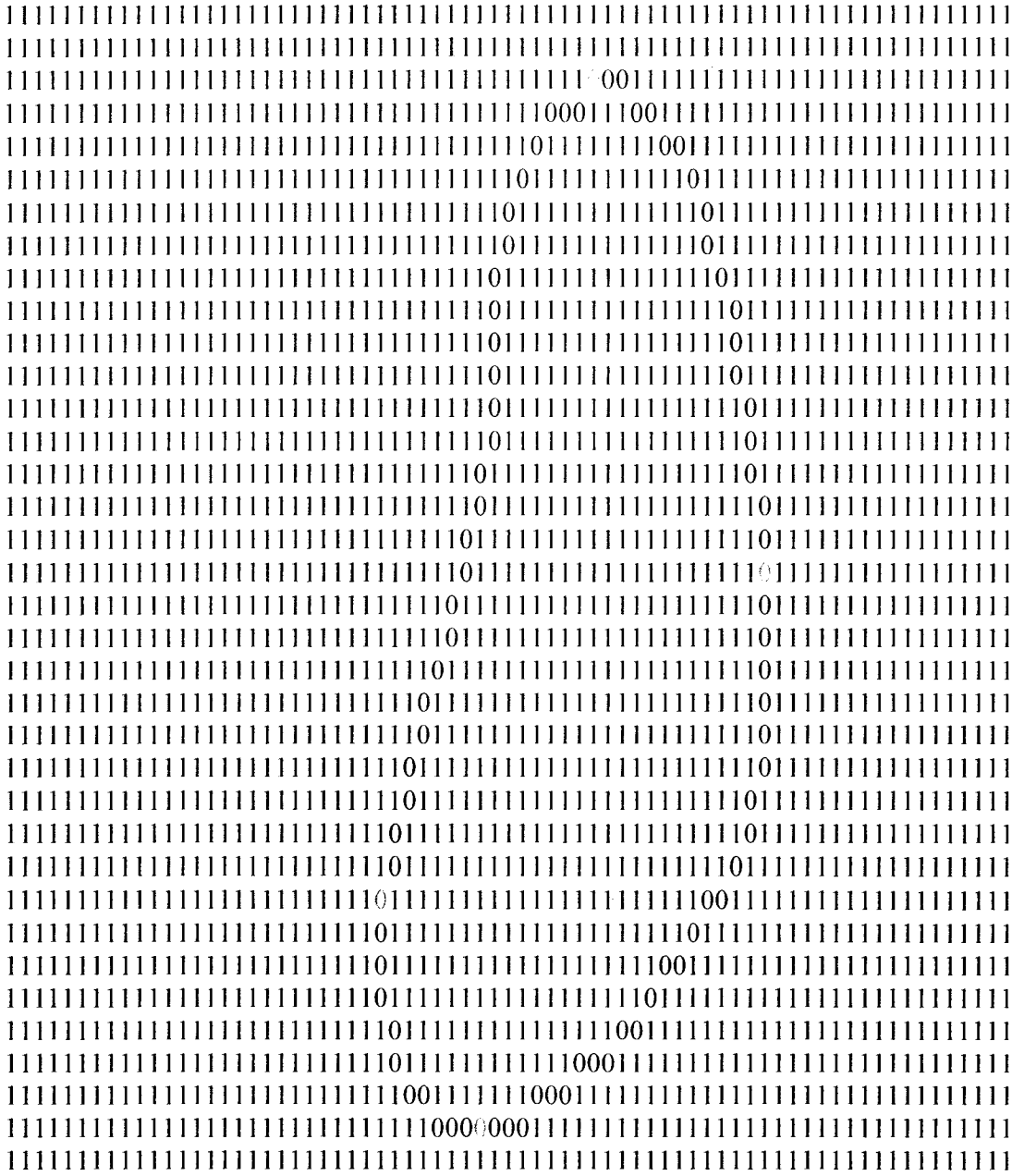


Figure 4.1 Extreme Points

Red color pixels show the extreme pixel position of the character that is present in the 128 by 128 pixels. With the help of these the position of the character can be find out and we can divide the character into block to find the local tips and total local tips.

4.1.2 Number of Tips

Using the thinning algorithm the image is first thinned. Once the image is thinned we will calculate the number of black pixels that are adjacent to each pixel. If the total number of pixels is one then we will identify it as tip of the character.

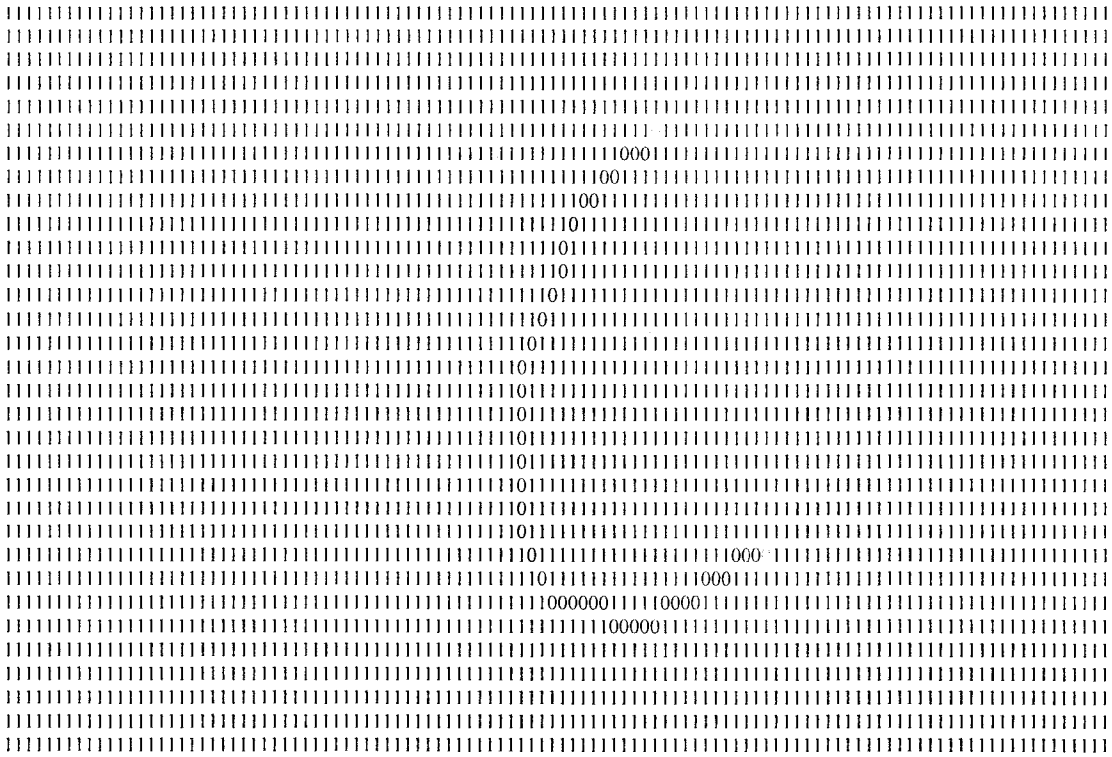


Figure 4.2 Number Tips

4.1.3 Change in Color of Pixels Horizontally

Using this algorithm we can find out the number of times the color of pixel changes from white to black at the mid point the character vertically.

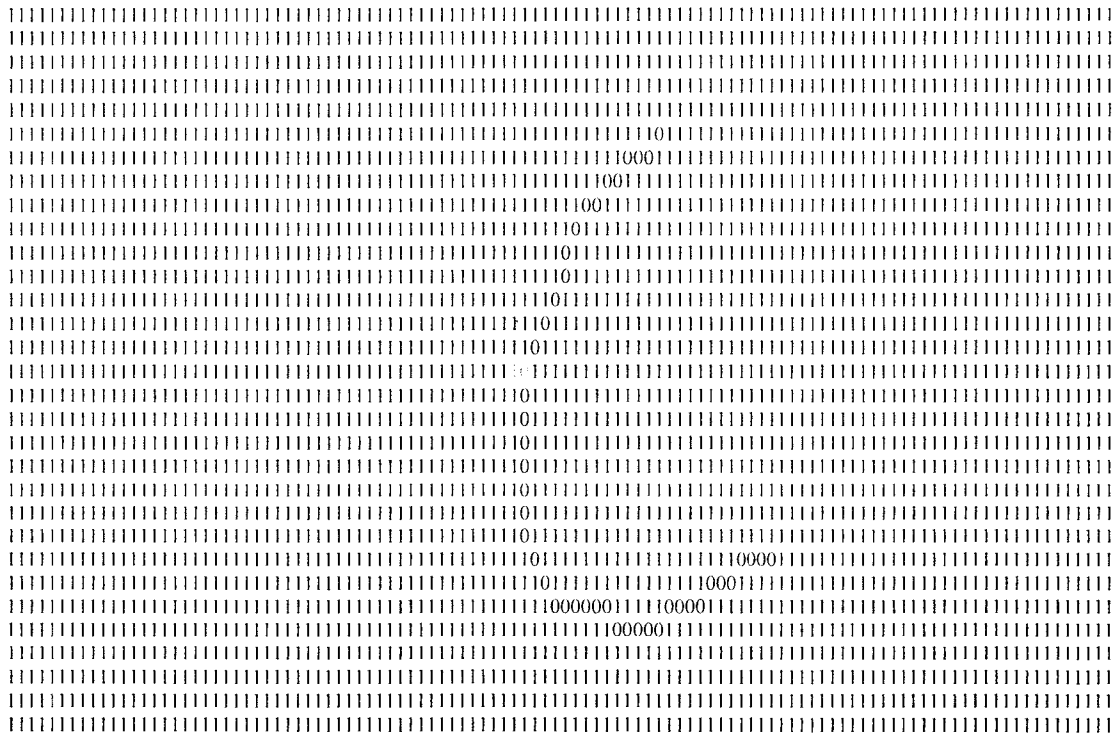


Figure 4.3 Change in color of pixels horizontally

The number of times the value of pixel changes from 1 to 0 at the mid point of the height of the character is 1.

4.1.4 Change in color of Pixels vertically

Using this algorithm we can find out the number of times the color of pixel changes from white to black at the mid point the character horizontally.

Example Character 'O'

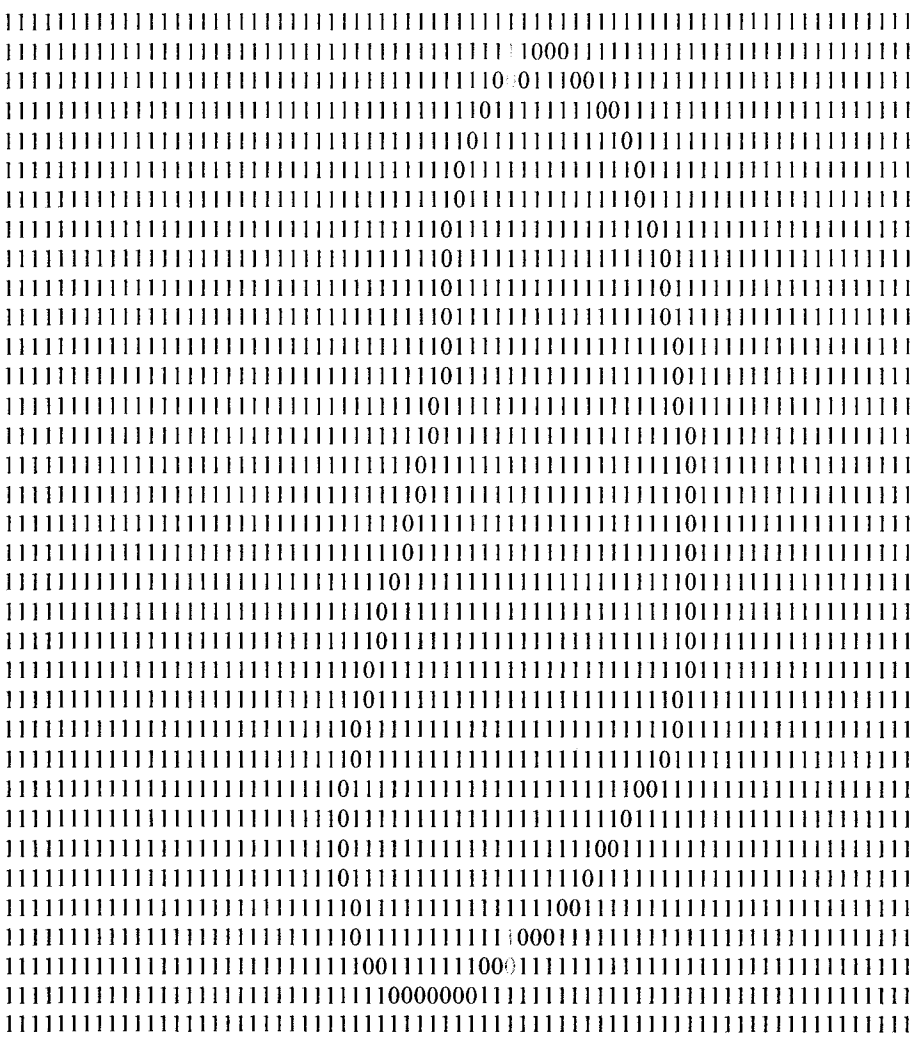


Figure 4.4 Change in color of pixels vertically

Here the total number of times the value of pixel changes from 1 to 0 at the mid point of the horizontal width of the character are 2.

4.2 Hierarchical Approach

The probability of an arbitrary character to occur in a random text location is about the same for each character, with the exception of special characters. Thus if we consider C characters (not counting special characters) that probability is approximately $P_i=1/C$, $I=1, 2, 3, \ldots, C$. The information is $I_i= \text{Log}_2(1/P_i) = \text{Log}_2(C)$, $i=1, 2, 3, \ldots..$, C and signifies the number of bits needed to encode a character. The entropy H of the system is $H = \sum_{i=1}^{C} P_i \text{ Log }_2(1/P_i) = \sum_{i=1}^{C} (1/C) \text{ Log }_2 C = \text{Log }_2 C$. The probability distribution function of the system is uniform, below we prove a theorem related to the expected number of searches associated with a set of items(Characters etc) having discrete uniform distribution .

Theorem1:

If a set S of C items has a uniform distribution for each item to occur, i.e. $P_i= 1/C$, $i=1, 2, \ldots..$, C and the entropy therefore is $\text{Log }_2 C$, then the expected number of searches needed to recognize a random item is C/2 , regardless of the operating order of the recognition algorithm.

Proof:

Let $j_1, j_2, \ldots..j_C$, $1 \leq Ji \leq C$, $i= 1,2,3,\ldots..C$ be the order which the pattern recognition algorithm follows in order to decide which one is the inputted item. The probability that the item will be recognized in the first trial is 1/C, in the second trial is $(C-1)/C^* 1/(C-1)$ and in the i^{th} trial is $((C-1)\ldots\ldots\ldots C-i+1)/(C^*(C-1)\ldots\ldots(C-i+1)) =1/C$ $I= 1, 2, 3\ldots C$. Hence the expected number of trials needed for one item to be recognized is $1/C+2^*1/C+\ldots\ldots+C^*1/C=1/C\ ^*(C(C+1))/2 = (C+1)/2$.

The feature vectors in character recognition contain a relatively large number of components. The pattern recognition algorithms associated with large feature vectors, are relatively slow due to the complexity of the operations. To speed up the recognition process first we use a very fast algorithm that although not very accurate, given that a certain character is scanned , they include this character as part of their recognition set and they also include some other characters with variable probability , while they exclude certain characters. The way this fast algorithm works is it recognizes that the character for example could be an 'A' or 'H' or a 'K' few other characters , but it could not be for example an 'O' or a 'I' etc.Associated with the recognition are probability having this information we rank the characters according to their probabilities and we employ the more accurate and more computationally expensive algorithm which recognizes the character much faster .To make our point for example if there are 10 possible characters in the set and the algorithm recognizes C_2 or C_5 or C_7 or C_8 only , and if any time the algorithm recognizes that the character is one of those characters then C_7 is the most likely with probability P_7 is the next likely with probability P_7, C_5 is the next likely with probability P_5 and C_8 is the last one with probability P_8.The expected number of comparisons for the comprehensive algorithm now is less than 2.5 as opposed to 5.5 that were before.

Given any set of characters to be recognized their number is finite , so it is straight forward to enumerate them with a finite sequence of numbers 1,2,3,.....,Max. After applying the recognition algorithm the set is reduced to $n_1, n_2, \ldots\ldots, n_n$ where $1 <= n_i <= Max$, i=1, 2, 3...., N. and N<Max. Further more if X is a random variable taking on the numbers $n_1, n_2, \ldots\ldots n_N$. $P(X=n_1) \geq P(X=n_2) \geq \ldots\ldots \geq P(X=n_N)$

From the experimental results using the capital handwritten letters of the NIST we obtained that the discrete probability function

$F(x)= e^{-ax}$, x=1,2,3,.....,N

Thus, $P(X=n_1) = e^{-1*a}$. $P(X=n_2)= e^{-2*a}$. $P(X=n_3)= e^{-3*a}$ ..... $P(X=n_N)= e^{-N*a}$

Since $P(X=n_1) +P(X=n_2)+.....+P(X=n_N)=1$

We have $e^{-1*a} + e^{-2*a} + e^{-3*a}+..... .....+ e^{-N*a} =1$

$=> e^{-a}(1+ e^{-1*a} + e^{-2*a} +..... e^{-(N-1)*a})=1$

$=> e^{-a}\{ (1- e^{-N*a})/ (1-e^{-a}) \} =1$

$=> 1- e^{-N*a}= e^{a} -1 => e^{-N*a} + e^{a} =2$. Using numerical methods we can compute a.

Theorem 2:

Using the preprocessing method and then the full scale character recognition algorithm the expected number of searches reduces from {Max+1}/2 to $e^{-a} \{(1- (N+1) e^{-N*a}+ Ne^{-(N+1)*a})/ (1-e^{-a})^2\}$ which for a<1 is approximately $e^{-a}/ (1-e^{-a})^2$

Proof :

The expected number of searches after the preprocessing is

$E(X)= e^{-a} + 2*e^{-2a} +3*e^{-3a} +..... + N*e^{-Na}$

Or $E(X)= e^{-a} (1+ 2*e^{-a} +3*(e^{-a})^2 +..... + N*(e^{-a})^{(N-1)})$

Or if $U=e^{-a}$ then $E(X)= U(1+ 2*U + 3*U^2 +.....+ N *U^{N-1})$

Or $E(X)= U\ d/du\ (1+U+U^2+.....+U^N)$

Or $E(X) =U\{(1-U^{N+1})/(1-U)\}= U \{-(N+1)U^N(1-U)+(1-U^{N+1})\}/(1-U)^2$

Or $E(X)= U \{1- U^{N+1}-NU^N +NU^{N+1}-U^N+U^{N+1})\}/ (1-U)^2 = U\{ 1-(N+1)U^N+NU^{N+1}\}/(1-U)^2$

$E(x)= e^{-a}\{ 1-(N+1) e^{-Na}+ Ne^{-(N+1)a}\}/(1-e^{-a})^2$

For $a<1$ this is approximately is $e^{-a}/(e^{a}-1)^{2}$ For the 26 capital English letters {Max+1}/2 =13.5 after preprocessing the expected numbers of searches are about 2.3.

CHAPTER 5

CLASSIFICATION AND IMPLEMENTATION

5.1 Classification

Once the preprocessing is done then comes the recognition of the character. But our objective is grouping the characters that have the same features and eliminate the character that does not have. This can be done by following the rules. Each character will have its own features that are not present in other characters. Hence these rules are used to classify each that have same features and avoid the other characters that do not have the same features.

Rules for character 'A'

1) It should have two tips at the bottom of character.
2) Total number of tips should not be more that 6.
3) When a horizontal line is drawn at the mid point vertically it should intersect two slant lines. Hence it should have two black pixels and the number of times the color of the pixel changes from 1 to 0 should be more than 2.

If the character that satisfies all the above conditions then it should be an 'A'. If the character does not satisfy then it is not an 'A'. When the following rules are applied to the input database character 'A' may be some times recognized as the other characters like 'R', 'M', 'W' since they have the same features described for the character 'A'. So 'M', 'R', 'W' fall into the group of having the same features as 'A'. But 'A 'may not

have the same features as characters like 'I' hence it doest not fall into the group. Even though the character 'A' may be recognized as other alphabets the probability would be different. If you take the 'R' since the features of 'A' and 'R' is same there is more probability that the alphabet 'A' may be recognized as 'R'. But if you take the character 'B' even though sometimes it may have features as 'A' the number of times the 'A ' recognized as 'B' will be comparatively less. Hence the probabability for 'A' to be recognized as 'B' will be less. If you take the characters like 'I', 'J' the probability that 'A' recognized as those character will be zero since the features of 'A' are completely different to those to the features of the 'I', 'J' .Hence we can increase the speed of the recognition by decreasing the number of comparisons by not comparing the characters whose probability is zero and apply the hierarchical approach for other characters whose probability is not zero. Hierarchical approach is which is been discussed in the previous chapter.

5.2 Implementation

The total number of characters that are used for testing the functioning of the algorithm is 2498 in HSF_2 database and 2498 for HSF_3 database.

Table 1 Input Database

HSF_2 characters

Number of A's	96
Number of B's	97
Number of C's	97
Number of D's	97
Number of E's	96
Number of F's	96
Number of G's	96
Number of H's	96
Number of I's	96
Number of J's	96
Number of K's	96
Number of L's	96
Number of M's	96
Number of N's	96
Number of O's	96
Number of P's	96
Number of Q's	96
Number of R's	96
Number of S's	96
Number of T's	96
Number of U's	96
Number of V's	95
Number of W's	96
Number of X's	96
Number of Y's	96
Number of Z's	96

HSF_3 Characters

Number of A's	95
Number of B's	97
Number of C's	97
Number of D's	97
Number of E's	96
Number of F's	96
Number of G's	96
Number of H's	96
Number of I's	96
Number of J's	96
Number of K's	96
Number of L's	96
Number of M's	96
Number of N's	96
Number of O's	96
Number of P's	96
Number of Q's	96
Number of R's	96
Number of S's	96
Number of T's	96
Number of U's	96
Number of V's	96
Number of W's	96
Number of X's	96
Number of Y's	96
Number of Z's	96

When the algorithm is ran the following results are been obtained for the character 'A'. The following table indicates that number of A's recognized as other characters. From the below table for HSF_2 we can under stand that the 10 times A's has been recognized as B. 18 times it been recognized as 'E'. 93, 96 , 42 times it been recognized as 'M', 'R', 'W' respectively .It is not recognized as D , I, J, L,O, S,T ,V,Y,Z.

Table 2 A's Recognized as Other Characters

DATABASE	HSF_2	HSF_3
Number of A's	93	89
Number of B's	10	9
Number of C's	2	1
Number of D's	0	3
Number of E's	18	12
Number of F's	21	15
Number of G's	4	4
Number of H's	8	6
Number of I's	0	0
Number of J's	0	0
Number of K's	9	6
Number of L's	0	0
Number of M's	93	89
Number of N's	23	22
Number of O's	0	0
Number of P's	2	3
Number of Q's	4	3
Number of R's	96	95
Number of S's	0	0
Number of T's	0	0
Number of U's	4	4
Number of V's	0	0
Number of W's	42	44
Number of X's	9	6
Number of Y's	0	0
Number of Z's	0	0

Table 3 Other Characters Recognized as A's

Output of HSF_2

Number of A's	93
Number of B's	13
Number of C's	0
Number of D's	4
Number of E's	11
Number of F's	0
Number of G's	0
Number of H's	90
Number of I's	0
Number of J's	0
Number of K's	61
Number of L's	2
Number of M's	89
Number of N's	48
Number of O's	2
Number of P's	1
Number of Q's	6
Number of R's	80
Number of S's	0
Number of T's	0
Number of U's	4
Number of V's	3
Number of W's	16
Number of X's	40
Number of Y's	2
Number of Z's	3

Output of HSF_3

Number of A's	89
Number of B's	13
Number of C's	0
Number of D's	5
Number of E's	9
Number of F's	0
Number of G's	1
Number of H's	88
Number of I's	0
Number of J's	0
Number of K's	74
Number of L's	0
Number of M's	90
Number of N's	60
Number of O's	2
Number of P's	3
Number of Q's	8
Number of R's	87
Number of S's	1
Number of T's	0
Number of U's	1
Number of V's	0
Number of W's	19
Number of X's	33
Number of Y's	2
Number of Z's	0

The following above table for database HSF_2 indicates that number of other characters recognized as 'A's. For example table indicates the out of the 96 'B's 13 are recognized as 'A''s .

CHAPTER 6

EFFIENCY AND CONCLUSION

6.1 Efficiency

If the general procedure is been followed then the each alphabet feature should be compared with all the other alphabet features of the input database. Even though the accuracy of the recognition process is more but the time consumption of the process is very high which is disadvantage for the recognition process. Hence the hierarchical approach is been followed where the number of comparisons required will be less.

Table 4 Probability of A's Recognized as Other Characters

	HSF_2	HSF_3
Probability of A's to be A's	0.212329	0.216545
Probability of A's to be B's	0.022831	0.021898
Probability of A's to be C's	0.004566	0.002433
Probability of A's to be D's	0.000000	0.007299
Probability of A's to be E's	0.041096	0.029197
Probability of A's to be F's	0.047945	0.036496
Probability of A's to be G's	0.009132	0.009732
Probability of A's to be H's	0.018265	0.014599
Probability of A's to be I's	0.000000	0.000000
Probability of A's to be J's	0.000000	0.000000
Probability of A's to be K's	0.020548	0.014599
Probability of A's to be L's	0.000000	0.000000
Probability of A's to be M's	0.212329	0.216545
Probability of A's to be N's	0.052511	0.053528
Probability of A's to be O's	0.000000	0.000000
Probability of A's to be P's	0.004566	0.007299
Probability of A's to be Q's	0.009132	0.007299
Probability of A's to be R's	0.219178	0.231144
Probability of A's to be S's	0.000000	0.000000
Probability of A's to be T's	0.000000	0.000000
Probability of A's to be U's	0.009132	0.009732
Probability of A's to be V's	0.000000	0.000000
Probability of A's to be W's	0.09589	0.107056
Probability of A's to be X's	0.020548	0.014599
Probability of A's to be Y's	0.000000	0.000000
Probability of A's to be Z's	0.000000	0.000000

From the output of the database HSF_2 obtained , the probability of the character 'A' to be recognized as the other characters we can understand that the features of the 'A' are

matching with the features of 'M','R', 'W' more compared to the features of the 'B','E', 'Q'. Hence the probabilities of A's recognized as 'M',' N','R','W' are more then compared to the probabilities of 'A' recognized as 'B','C','D','E'. Features of other alphabets such as 'D','I', 'J', 'L', 'O' ,'S', 'T', 'V', 'Y', 'Z' are completely contrast hence the probability of A's to be recognized is zero. Since the probability that the 'A' recognized as other characters is not the same the hierarchical approach is been followed. Since there are total of 26 characters present in the input database the total number of comparisons required will be (26+1)/2 Which is equal to 13.5.But if the number the hierarchical approach is been followed the number of comparisons will be less which is been showed below. Total number of characters is 26.The number of characters whose probability is zero is 10. Hence the number characters that 'A' should be compared to classify is 26-10=16.Hence the number of comparisons required is (16+1)/2= 17/2=8.5.

From the above results we can understand that the number of comparisons required is less. Since the probability that the 'A' recognized as other characters is not same for all the characters. Therefore the hierarchical approach is followed where the characters that have the maximum probability is compared first and which have the minimum probability is compared last. Hence the characters are ranked in the order of their probability.

Table 5 Rank of Characters

	HSF_2	Rank
Probability of A's to be R's	0.219178	1
Probability of A's to be A's	0.212329	2
Probability of A's to be M's	0.212329	3
Probability of A's to be W's	0.09589	4
Probability of A's to be N's	0.052511	5
Probability of A's to be F's	0.047945	6
Probability of A's to be E's	0.041096	7
Probability of A's to be B 's	0.022831	8
Probability of A's to be K's	0.020548	9
Probability of A's to be X's	0.020548	10
Probability of A's to be H's	0.018265	11
Probability of A's to be G's	0.009132	12
Probability of A's to be Q's	0.009132	13
Probability of A's to be U's	0.009132	14
Probability of A's to be C's	0.004566	15
Probability of A's to be P's	0.004566	16
Probability of A's to be D's	0.000000	17
Probability of A's to be I's	0.000000	18
Probability of A's to be J's	0.000000	19
Probability of A's to be L's	0.000000	20
Probability of A's to be O's	0.000000	21
Probability of A's to be S's	0.000000	22
Probability of A's to be T's	0.000000	23
Probability of A's to be V's	0.000000	24
Probability of A's to be Y's	0.000000	25
Probability of A's to be Z's	0.000000	26

For the database HSF_2 we can see the ranks of the characters that 'A' been recognized as others. Hence from the table we can understand that 'A' has to be

compared with either 'A', 'M', 'R' and then with 'W' ,'N', 'E' and so on. The sum of the probabilities should be equal

$=> P_A+P_B+P_C+P_D+P_E+P_F+P_G+P_H+P_I+P_J+P_K+P_L+P_M+P_N+P_O+P_P+P_Q+P_R+P_S$

$+P_T+P_U+P_V+P_W+P_X+P_Y+P_Z = 1$

$=> P_A(1+(P_B+P_C+P_D+P_E+P_F+P_G+P_H+P_I+P_J+P_K+P_L+P_M+P_N+P_O+P_P+P_Q+P_R+P_S$

$+P_T+P_U+P_V+P_W+P_X+P_Y+P_Z)/P_A) = 1.$

$=> P_A= 1/((1+(P_B+P_C+P_D+P_E+P_F+P_G+P_H+P_I+P_J+P_K+P_L+P_M+P_N+P_O+P_P+P_Q$

$+P_R+P_S+P_T+P_U+P_V+P_W+P_X+P_Y+P_Z)/P_A).$

Expected number of trials is

$P_A*R_A+P_B*R_B+P_C*Rc+P_D*R_D+P_E*R_E+P_F*R_F+P_G*R_G+P_H*R_H+P_I*R_I+P_J*R_J$

$+P_K*R_K+P_L*R_L+P_M*R_M+P_N*R_N+P_O*R_O+P_P*R_P+P_Q*R_Q+P_R*R_R+P_S*R_S+P_T*R_T$

$+P_U*R_U+P_V*R_V+P_W*R_W+P_X*R_X+P_Y*R_Y+P_Z*R_Z$

Table 6 Expected Number of Trials

	HSF_2	Rank	Rank* Probability
Probability of A's to be R's	0.219178	1	0.219178
Probability of A's to be A's	0.212329	2	0.424658
Probability of A's to be M's	0.212329	3	0.636987
Probability of A's to be W's	0.09589	4	0.38356
Probability of A's to be N's	0.052511	5	0.262555
Probability of A's to be F's	0.047945	6	0.28767
Probability of A's to be E's	0.041096	7	0.287672
Probability of A's to be B 's	0.022831	8	0.205479
Probability of A's to be K's	0.020548	9	0.20548
Probability of A's to be X's	0.020548	10	0.226028
Probability of A's to be H's	0.018265	11	0.21918
Probability of A's to be G's	0.009132	12	0.118716
Probability of A's to be Q's	0.009132	13	0.127848
Probability of A's to be U's	0.009132	14	0.13698
Probability of A's to be C's	0.004566	15	0.073056
Probability of A's to be P's	0.004566	16	0.077622
Probability of A's to be D's	0.000000	17	0
Probability of A's to be I's	0.000000	18	0
Probability of A's to be J's	0.000000	19	0
Probability of A's to be L's	0.000000	20	0
Probability of A's to be O's	0.000000	21	0
Probability of A's to be S's	0.000000	22	0
Probability of A's to be T's	0.000000	23	0
Probability of A's to be V's	0.000000	24	0
Probability of A's to be Y's	0.000000	25	0
Probability of A's to be Z's	0.000000	26	0

The total expected number of trials required for database HSF_2 is 3.892669 which is less compared to 8.5.The same procedure is been applied to all the characters in the

database which shows that the number of comparisons required for classifications of the characters is much less if the hierarchical procedure is been followed. There by reducing the number of comparisons we are able to save time for the recognition process.

6.2 Conclusion

Optical character recognition is way in which the printed or handwritten text is converted into machine editable form. Character recognition is very important in modern day life where it is used in the postal services, banking and insurance companies and many more. Hence it is very important field and the research is going on in this field. Recognition of the hand written characters is very difficult since each user writes the same character in different shape and size. In order to recognize the characters following procedure has to be followed.

1) Character should be converted to binary image
2) Big dots that are present in the image should be removed.
3) Noise that is present in the image has to be removed.

Once the preprocessing has been done then it has to be followed by the feature extraction. Once the features are extracted then the resultant features must be trained and tested by using the classifier. There are different classifiers that can be used in the optical character recognizers.

They are

1) Statistical Methods
2) Artificial Neural Network Method.
3) Kernel Method.

4) Combination of Multiple Classifiers.

The above methods will help in high accuracy .But the time taken will be more which is a disadvantage for the recognizer since it has to be trained and tested against huge data base of characters. Hence the hierarchical approach is been followed to increase the efficiency by decreasing the number of comparisons by the recognizer to classify .Since all characters would not have the same features hence the characters which not similar can be eliminated for comparison. There by reducing the number of comparisons required by the classifier to classify the characters. After eliminating the characters whose features are not same, then the other characters are compared in the order of their probabilities. There by we can increase the speed of the recognition process.

REFERENCES

1. C.L.Wilson and M.D.Garris. Handprinted character database. Technical Report Special Database 1, HWDB, National Institute of Standards and Technology, April 1990.
2. R.A Wilkinson. Hand printed Segmented Characters Database. Technical Report Test Database 1, TST1, National Institute of Standards and Technology, February 1992.
3. Meenakshisundaram Murugan. "Handwritten Number Recognition". MS thesis. University of Nevada , Las Vegas .May 2005.
4. Mageshkumar Padmanaban . "Handwritten Character Recognition by Combining Multiple Recognizers Using Conditional Probabilities". MS Thesis ,University of Nevada Las Vegas. December 2005.
5. Pu Han et al. "Car license plate feature extraction and recognition based on multi stage classifier". Proceeding of Second International Conference on Machine learning and Cybernetics, Xi'an,2-5 November 2003.
6. Liangrui Pen et al. "Multilingual Document recognition research and its applications in China". Proceeding of Second International Conference on Document Image Analysis for Libraries(DIAL 2006)
7. P.J. Grother. Hand printed forms and character database, NIST special database19,March 1995. Technical Report and CDROM.

8. C.Y. Suen, J. Tan, Analysis of error of handwritten digits made by a multitude of classifiers, *Pattern Recognition Letters*, 26(3): 369-379, 2005

9. A. F.R. Rahman, M.C. Fairhurst, Multiple classifier decision combination strategies for character recognition: a review, *Int. J. Document Analysis and Recognition*, 5(4): 166-194, 2003.

10. L. Xu, A. Krzyzak, C.Y. Suen, Methods of combining multiple classifiers and their applications to handwriting recognition, *IEEE Trans. System Man Cybernet.*, 22(3): 418-435, 1992.

11. J. Kittler, M. Hatef, R.P.W. Duin, J. Matas, On combining classifiers, *IEEE Trans. Pattern Anal. Mach.Intell.*, 20(3): 226-239, 1998.

12. V. Vapnik, *The Nature of Statistical Learning Theory*,Springer-Verlag, New Work, 1995.

13. C.J.C. Burges. A tutorial on support vector machines for pattern recognition, *Knowledge Discovery and Data Mining*, 2(2): 1-43, 1998.

14. A.F.R. Rahman, M.C. Fairhurst, Multiple classier decision combination strategies for character recognition: a review, *Int. J. Document Analysis and Recognition*, 5(4):166-194, 2003.

15. F.Kimura, K.Takashima, S.Tsuruoka and Y.Miyake "Modified Quadtratic discriminant functions and the application to Chinese character recognition", IEEETrans Pattern Anal. Machine Intell., vol.PAMI-9, pp149-153, Jan.1987.

16. C.J Hilditch, "Linear skeletons from square cupboards", Machine Intelligence 4, (Edinburgh Univ,Press), pp 403-420

17. M. Blumenstein and B. Verma, "Neural–based solutions for the segmentation and recognition of difficult handwritten words from a benchmark database", Proc. 5th International Conference on Document Analysis and Recognition, pages 281–284, Bangalore, India, 1999.

18. P. D. Gader, M. A. Mohamed, and J. H. Chiang, "Handwritten word recognition with character and inter–character neural networks", IEEE Transactions on Systems, Man and Cybernetics – Part B, 27:158–164, 1994

APPENDIX

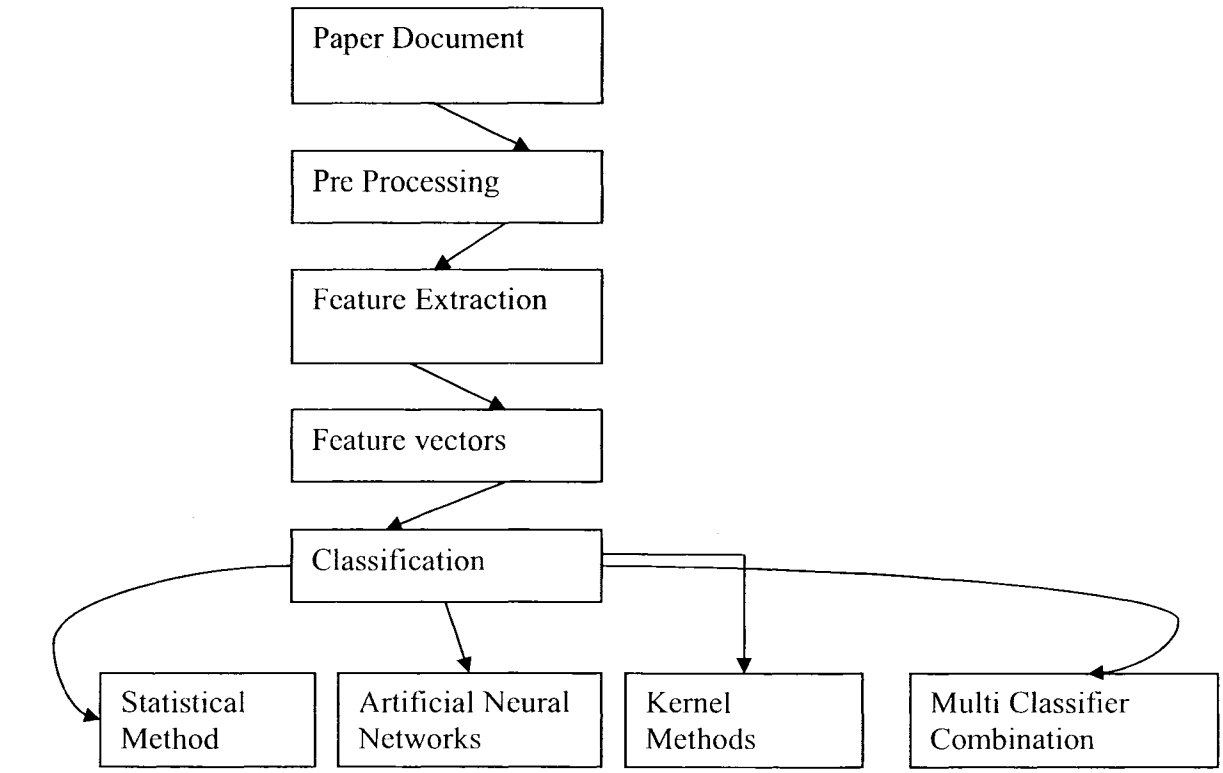


Figure 1 General Approach

In the general approach the paper document where the user writes the text will be scanned using scanner. Once the paper is been digitized then the preprocessing will be implemented to increase the accuracy and efficiency of the algorithm. Once the preprocessing is done the features will be extracted from the characters and stored in the feature vectors .All these feature vectors will be used by the classifier to recognized the

characters. Since each character has to be compared with all other characters the time taken will be less. To increase the speed of the recognition the hierarchical approach is been followed.

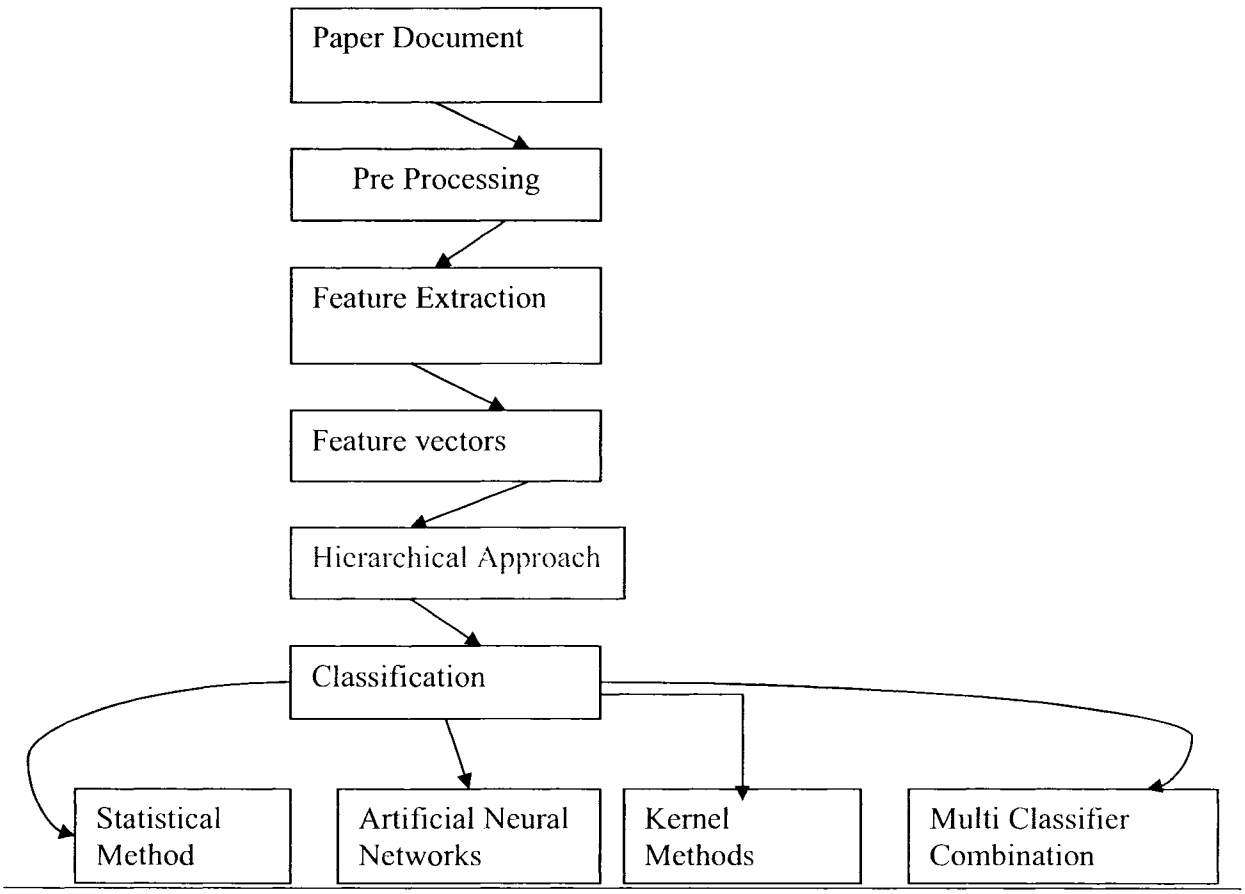


Figure 2 Hierarchical Approach

In the above approach after the feature vectors are extracted the hierarchical approach will be implemented where by the number of characters that are to be compared will be less then the maximum number of characters that are present in the characters.

Functions used in the Implementation

1. Hilditch Thinning algorithm

2. ConnectedComponentSementer
3. FilterBySize
4. Findminmax
5. Number of tips

1)Hilditch Thinning algorithm

Thinning is process in which object image width will be reduced to one pixel width.

Functions used in the thinning algorithm.

1.1)calculateNZ

This function will be calculating the number of black pixels that are adjacent to it

1.2) calculateNT

This function will be calculating the number of times the value of the pixel color changes 1 to 0 in the pattern p2, p3 ,p4,p5,p6,p7,p8,p9,p2.

2)FilterbySize

This function is used to remove the speckles that are present in the character image.

3) ConnectedComponentSegmenter

This function will be help you to count the number of components that are present in the image.

2)Findminmax

This function will find out the size and the position of the character that is present in the 128 by 128 image.

3)Number of Tips

This function will find of the number of tips present in the thinned image.

1. Hilditch Function

```
package edu.unlv.cs.ip;

public class TwoPassThinner implements Thinner {


   public ImageInterface apply(final ImageInterface image)
         throws ImageProcessingException {
      return hilditch(image);
   }



public ImageInterface hilditch( ImageInterface image )

{

     int treshold=1;
     int  []points=new int[10];

     int loc;
     int image_size;
     int image_row, image_col;
     boolean changed=true;
     int i, j;
     char option;


     image_row =image.getHeight();
     image_col = image.getWidth();
     image_size = image_row * image_col;
     int []image_original = new  int[image_size];

     ImageInterface image2=image.copy();

     for(int r=0;r<128;r++)
          for(int p=0;p<128;p++)
          {
               if(image2.getPixel(r,p)==1)
                    image_original[p+r*128]=1;
               else
                    image_original[p+r*128]=0;
          }

     for( i=0; i<image_col; i++){
          for( j=0; j<image_row; j++)

     {
          if( image_original[i+j*128] >= 1 )
               image.setPixel(j,i,0); //set black
        else
               image.setPixel(j,i,1); //set white
          }
     }
```

```
int []     m1 = new int[image_size];
int []     m2 = new int[image_size];
int []     m0 = new int[image_size];
     for( i=0; i<image_size; i++){

          if(image_original[i]==0)
                    {
               m1[i] = 1; //set black
               m2[i] = 1;
               m0[i] = 1;
          }
          else{
               m1[i] = 0; //set white
               m2[i] = 0;
               m0[i] = 0;
          }
     }

     // 4 thinning conditions
     //cout << "\nApplying thinning.....\n";
     while(changed){

               changed = false;

               //set A - delete South & West bounds
               for(int x=1; x<image_row-1; x++){
                    i = x*128;
                    for(int y=1; y<image_col-1; y++){
                         j = y;

                         //fill 3x3 points using m1 at (i,j)
                         fillPoints(i,j,m1,points,image_col);

                         if(points[1] == 1){     // p1 == 1
     /*cond1*/                 if( calculateNZ(points) >= 2 ){
     /*cond2*/                      if( calculateNT(points) == 1 ){
     /*cond3*/                           if( (points[2] * points[4] * points[6]) == 0 ){
     /*cond4*/                                if( (points[2] * points[4] * points[8]) == 0 ){
                                                 m2[i+j] = 0; //set white (delete)
                                                 changed = true;
                                            }//end if cond4
                                       }//end if cond3
                                  }//end if cond2
                             }//end if cond1
                         }//end if p1 == 1
                    }//end for y
               }//end for x


               // copy matrix to for set B
               if(changed){
                    for(int n=0; n<image_size; n++)

                         m1[n] = m2[n];
               }
```

```
            //set B - delete North & East bounds
            for(int x=1; x<image_row-1; x++){
                i = x*128;
                for(int y=1; y<image_col-1; y++){
                    j = y;

                    //fill 3x3 points using m1 at (i,j)
                    fillPoints(i,j,m1,points,image_col);

                    if(points[1] == 1){     // p1 == 1
/*cond1*/               if( calculateNZ(points) >= 2){
/*cond2*/                   if( calculateNT(points) == 1 ){
/*cond3*/                   if( (points[2] * points[6] * points[8]) == 0 ){
/*cond4*/                   if( (points[4] * points[6] * points[8]) == 0 ){
                                m2[i+j] = 0; //set white (delete)
                                        changed = true;
                                    }//end if cond4
                                }//end if cond3
                            }//end if cond2
                        }//end if cond1
                    }//end if p1 == 1

                }//end for y
            }//end for x

            // copy matrix to final
            if(changed){
                for(int n=0; n<image_size; n++)

                    m1[n] = m2[n];
            }


        }//end while


        //clean up
        for(int x=1; x<image_row-1; x++){
            i = x*128;
            for(int y=1; y<image_col-1; y++){
                j = y;
                if( m1[i+j] > 0){
                    fillPoints(i,j,m1,points,image_col);
                    if(calculateNT(points) == 2){                    //means 2-way split
if(points[2]+points[4]==2 || points[4]+points[6]==2 ||                   points[6]+points[8]==2 ||
points[8]+points[2]==2){
                            m1[i+j] = 0;                             //extra, delete point
                            m2[i+j] = 0;
                        }
                    }
                    else if(calculateNT(points) == 3){
                        if(points[2]+points[4]+points[6]==3 || points[4]+points[6]+points[8]==3 ||
                          points[6]+points[8]+points[2]==3 || points[8]+points[2]+points[4]==3){
                            m1[i+j] = 0;
```

```
                                m2[i+j] = 0;
                            }
                        }
                    }
                }
            }
            //(((up to now, m0=original(0|1), m1=m2=thinned&cleaned))))

            for(int r=0; r<image_size; r++){
                if( m1[r] == 1 )
                    image_original[r] = 0; //set black
                else
                    image_original[r] = 1; //set white
            }

        // convert imagematrix(0|1) back to image_data(.bmp)
        for( i=0; i<image_col; i++){
            for( j=0; j<image_row; j++)

            if( image_original[j+i*128] == 1 )
                image.setPixel(j,i,1); //set black
            else
                image.setPixel(j,i,0); //set white
        }



        return image;
    }
    public void fillPoints(int i, int j, int []image_matrix,
                                        int []points, int c)
    {
        // p5 p6 p7
        // p4 p1 p8
        // p3 p2 p9

        points[1] = image_matrix[(i  )+(j  )];
        points[2] = image_matrix[(i+c)+(j  )];
        points[3] = image_matrix[(i+c)+(j-1)];
        points[4] = image_matrix[i +j-1];
        points[5] = image_matrix[(i-c)+(j-1)];
        points[6] = image_matrix[(i-c)+(j  )];
        points[7] = image_matrix[(i-c)+(j+1)];
        points[8] = image_matrix[(i  )+(j+1)];
        points[9] = image_matrix[(i+c)+(j+1)];

    }

    public int calculateNZ(int []pt)
    {
        // non-zero neighbors to p1

        return (pt[2]+pt[3]+pt[4]+pt[5]+pt[6]+pt[7]+pt[8]+pt[9]);
    }
```

```
public int calculateNT(int []pt)
{
    // zero(0) to one(1) transition
    // order = {p2,p3,p4,p5,p6,p7,p8,p9,p2}

    int transition=0;

    if( (pt[3]-pt[2]) == 1 ) transition++;
    if( (pt[4]-pt[3]) == 1 ) transition++;
    if( (pt[5]-pt[4]) == 1 ) transition++;
    if( (pt[6]-pt[5]) == 1 ) transition++;
    if( (pt[7]-pt[6]) == 1 ) transition++;
    if( (pt[8]-pt[7]) == 1 ) transition++;
    if( (pt[9]-pt[8]) == 1 ) transition++;
    if( (pt[2]-pt[9]) == 1 ) transition++;


    return transition;
}



}
```

2. Grouping Function

```
package edu.unlv.cs.ocr.structural;

import java.awt.Rectangle;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import org.animism.math.Statistics;

import edu.unlv.cs.ip.BinaryConverter;
import edu.unlv.cs.ip.BinaryImage;
import edu.unlv.cs.ip.DirectImage;
import edu.unlv.cs.ip.GrayscaleImage;
import edu.unlv.cs.ip.ImageInterface;
import edu.unlv.cs.ip.ImageProcessingException;
import edu.unlv.cs.ip.ImageRotator;
import edu.unlv.cs.ip.Thinner;
import edu.unlv.cs.ip.TwoPassThinner;
import edu.unlv.cs.ip.filter.GaussianFilter;
import edu.unlv.cs.ocr.CharacterLibrary;
import edu.unlv.cs.ocr.ConnectedComponent;
import edu.unlv.cs.ocr.ConnectedComponentSegmenter;
```

```
import edu.unlv.cs.ocr.*;

/**
 *
 *
 */
public class RegUsingEdge extends CharacterLibrary{

    protected ImageInterface thin;
    protected ImageInterface
m_img,m_binaryImage,m_binaryImage2,m_imageOne,m_imageTwo,m_imageThree,thinned_ima
ge,thinned_image2,shadow1_image,shadow2_image,shadow3_image;
    protected ImageInterface m_img1,thinned,thinned1;
    protected Thinner m_thinner = new TwoPassThinner();
    protected Thinner m_thinner1 = new TwoPassThinner();
    int[][] imagearray=new int[128][128];
    int[][] imagearray2=new int[128][128];
    int[][] imagesmooth=new int[128][128];
    int[][] imagesmooth2=new int[128][128];
    int [][] imagecopy= new int[132][132];
    GaussianFilter gf = new GaussianFilter();

    protected CharacterSeparator m_csParent;
    protected ColorImageThinning m_citParent;
    int r1=0;

    int xlmin,xlmax;
    protected ConnectedComponentSegmenter m_segmenter = new
ConnectedComponentSegmenter();
    protected LinkedList m_segList = null;
    int xmax,xmin;
    static int TotHeight,TotWidth;
    int[] RotX1=new int[8];
    int[] RotY1=new int[8];



    int lmax,lmin;

    int Xmin=200,Ymin=200,Xmax=0,Ymax=0;
    int RegGrp=0;
    int temp1;

    int []ActualCharGroup=new int[26];
    int [][]ActualChar=new int[26][500];
    int [][]outputchar=new int[26][500];


    int [] subclass=new int[1000];

    int label=0;
    int charcnt=0;
    int images=0,labels=0;
    int rows=0,cols=0;
    int maxX=0,maxY=0,minX=0,minY=0;
```

```
int parts=3;
int features=34;

int CGx=0,CGy=0;

int EXmin=0,EYmin=0,EXmax=0,EYmax=0;
int LEXmin=0,LEYmin=0,LEXmax=0,LEYmax=0;
int REXmin=0,REYmin=0,REXmax=0,REYmax=0;



double [][][]ResVec=new double [26][26][parts*(features)];
int []sort=new int[10];

int[] tippositionx=new int[100];
int[] tippositiony=new int [100];

 int[]junk= new int[26];

int Zeros=0,Ones=0;

public static final int TRAINING_SET = 0;
public static final int TEST_SET = 1;

protected static final int IMAGE_MAGIC_NUMBER = 2051;
protected static final int LABEL_MAGIC_NUMBER = 2049;
    protected String m_imageFile;
protected String m_labelFile;
protected boolean m_convertToBinary = true;
protected String m_directory;
protected boolean m_isLoaded = false;

protected int m_threshold = 128;

protected DigitRecognizer m_recognizer = new DigitRecognizer();

private List filterBySize(List components) {
    double[] areas = new double[components.size()];
    ArrayList out = new ArrayList();
    for (int i=0;i<areas.length;i++) {
       ImageComponent component = (ImageComponent) components.get(i);
       areas[i] = component.height()*component.width();
    }
    double std = Statistics.standardDeviation(areas);
    double mean = Statistics.average(areas);
   // m_logger.info("mean = "+mean+", std = "+std);

    for (int i=0;i<components.size();i++) {
       ImageComponent component = (ImageComponent) components.get(i);
       if (component.height()*component.width()<mean-std/2)  {
          out.add(component);
       }
    }
    return out;
 }
```

```
    private void remove(ImageInterface original, ConnectedComponent component) {
        for (Iterator i = component.getPixels().iterator();
        i.hasNext();) {
            Point p = (Point)i.next();
            original.setPixel(p.x,p.y,original.getBackgroundColor());
        }
     }



    public RegUsingEdge(ImageInterface image){
        this(TRAINING_SET);
        m_img = image;
    }

    public RegUsingEdge(int mode)
    {
        if ((mode % 2) == TRAINING_SET) {
            m_imageFile = "C:\\eclipse\\database\\HSF_2\\Test.txt";//*/TRAINING_FILE;
            m_labelFile = "C:\\eclipse\\database\\HSF_2\\TestLabel.txt";//*/TRAINING_LABELS;
          } else {
            m_imageFile ="C:\\eclipse\\database\\HSF_2\\Test.txt";// TEST_FILE;
            m_labelFile ="C:\\eclipse\\database\\HSF_2\\TestLabel.txt";//TEST_LABELS;
          }
    }

    public void setDirectory(String directory) {
            m_directory = directory;
        }

        public void setConvertToBinary(boolean convert) {
            m_convertToBinary = convert;
        }

    /**
     * Restore image info from persistant file
     * The data comes with two files:
     * One containing the image data and one the labels
     * @throws IOException
     * @throws ImageProcessingException
     */
    public void restoreData(double [][][][]P) throws IOException, ImageProcessingException {

restoreData("C:\\eclipse\\database\\HSF_2\\Test.txt","C:\\eclipse\\database\\HSF_2\\TestLabel.txt
",P);
        }


    public void restoreData(String imageFile, String labelFile,double [][][][] P)throws IOException,
ImageProcessingException{

        int m=0,n=0;

        if (m_isLoaded)
         {
```

```
            m_logger.info("Data is already loaded.");
            return;
          }

           System.out.println("asdaslkn");

           FileInputStream imageStream = new FileInputStream(imageFile);//open a stream for
reading the image file
           DataInputStream is = new DataInputStream(imageStream);

           FileInputStream labelStream = new FileInputStream(labelFile);// open a stream for
reading the labels file
           DataInputStream ls = new DataInputStream(labelStream);

           // read number of images and labels (should match)
          images = is.readInt();
          labels = ls.readInt();

           // read number of rows and columns
           int rows = is.readInt();
           m_logger.info("Rows = "+rows);
           int cols = is.readInt();
           m_logger.info("Columns = "+cols);
           images = Math.min(images, m_maxImages);
           m_logger.info("Reading " + images + " images.");
           m_logger.info("m_maxImages : "+m_maxImages);
           m_imageTwo = new DirectImage(cols,rows,DirectImage.TYPE_BINARY);
           m_imageThree = new DirectImage(cols,rows,DirectImage.TYPE_BINARY);
           shadow1_image = new DirectImage(cols,rows,DirectImage.TYPE_BINARY);
           shadow2_image = new DirectImage(cols,rows,DirectImage.TYPE_BINARY);
           shadow3_image = new DirectImage(cols,rows,DirectImage.TYPE_BINARY);

           // this loop reads each image byte by byte and stores in an array
          for (int p = 0; p < images; p++) {
           m_binaryImage = new DirectImage(cols, rows, DirectImage.TYPE_BINARY);

           System.out.println(p);
           label = (int) ls.readInt();//            read the label
               System.out.println("label : "+label);
               char c = Integer.toString(label).charAt(0);



          for (int j = 0; j < rows; j++)
               for (int i = 0; i < cols; i++) {

                    int value = is.readByte();

                    m_binaryImage.setPixel(i, j,(value==0)?1:0);                    //else
                        // ;

               }
```

```
//          Removing Spurts and big dots from the binary image
            m_segList = null;
            m_segList = (LinkedList) m_segmenter.segment(m_binaryImage);

            System.out.println("# of components: " + m_segList.size());

            int Area = 0;
            int maxArea = 0;
            int CharList = 0;
            ConnectedComponent comp = null;

            for (int i = 0; i < m_segList.size(); i++) {
                comp = (ConnectedComponent) m_segList.get(i);

                Area = (comp.maxX - comp.minX) * (comp.maxY - comp.minY);
                maxX=comp.maxX;
                maxY=comp.maxY;
                minX=comp.minX;
                minY=comp.minY;

                if (Area > maxArea) {
                    maxArea = Area;
                    CharList = i;
                }
            }

            ConnectedComponent cc = (ConnectedComponent) m_segList.get(CharList);
            Rectangle rect = cc.getEnclosingRectangle();



            m_imageOne = m_binaryImage.copy();

            //TODO Remove speckles
            try {
        ImageInterface binary = new BinaryConverter(200).apply(m_imageOne);
        List components = m_segmenter.segment(binary);
        List out = filterBySize(components);
        for (int i=0;i<out.size();i++) {
            ConnectedComponent component = (ConnectedComponent) out.get(i);
            remove(m_imageOne,component);
        }

      } catch (ImageProcessingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }



    for (int j = 0; j < rows; j++)
    {
        for (int i = 0; i < cols; i++)
        {
    imagecopy[i+2][j+2]=m_imageOne.getPixel(i,j);
        }
```

```
}
for (int j = 2; j < rows+2; j++)
{
    imagecopy[1][j]=imagecopy[3][j];
    imagecopy[0][j]=imagecopy[4][j];
    imagecopy[130][j]=imagecopy[128][j];
    imagecopy[131][j]=imagecopy[127][j];
}

for (int j = 0; j < cols+4; j++)
{
    imagecopy[j][1]=imagecopy[j][3];
    imagecopy[j][0]=imagecopy[j][4];
    imagecopy[j][130]=imagecopy[j][128];
    imagecopy[j][131]=imagecopy[j][127];
}



for (int i = 1; i < rows+1; i++)
     for (int j = 1; j < cols+1; j++)
     {

         {

  imagesmooth2[j-1][i-1]=(int) (((double)1/16)*((imagecopy[j-1][i-1]+2*imagecopy[j-
1][i]+imagecopy[j-1][i+1])
                                  +(2*imagecopy[j][i-1]+4*imagecopy[j][i]+2*imagecopy[j][i+1])
                                                  +(imagecopy[j+1][i-
1]+2*imagecopy[j+1][i]+imagecopy[j+1][i+1])));

         }
     }

  for(int i = 0; i < m_imageOne.getWidth(); i++)
          for(int j = 0; j < m_imageOne.getHeight(); j++)
          {


              imagearray[i][j]=imagesmooth2[i][j];//m_imageOne.getPixel(i,j);

          }




          m_imageTwo=m_imageOne.copy();

         //hsf_2 p==104||p==1711||p==358
         //hsf_3 p==546||p==156
          int temp10=0;


          if(  p==104||p==1711||p==358)
          {
```

```
                    temp10=1;
            }
                    else
                    {
                    temp10=0;
NumberOfTips2(imagearray);
for(int f=0;f<m_imageOne.getWidth();f++)
{
    for(int f1=0;f1<m_imageOne.getHeight();f1++)
    {
        m_imageOne.setPixel(f1,f,imagearray[f1][f]);
    }


}


Findmaxmin(0,0,128,128);
System.out.println("xmin"+minX+"  xmax"+maxX+"   ymin"+minY+"ymax  "+maxY);
int total=NumberOfTips(0,0,128,128);

int ymax=0;
int ymin=200;

int []sy=new int[total];
int []sx=new int[total];
for(int i=0;i<total;i++)
{
sy[i]=tippositiony[i];
}



if(total>1)
{
lmax=sy[total-1];//ymax
lmin=sy[total-2];//ysecond max
xlmin=tippositionx[total-2];//xsecondmax
xlmax=tippositionx[total-1];//xmax
}


thinned=m_thinner.apply(m_imageOne);//thinning

    int r=0;

int u1=0;
int u2=0;
int u3=0;
int xtop=0;
int xbottom=0;
int xleft=0;
int xright=0;
int temp=0;
int y1=0;
int y2=0;
```

```
int y3=0;
int y4=0;
int pix=0;
int b=0;
int pixels=0;
int pixelc=0;
int pixelt=0;
int F1=0;
int F2=0;
int F3=0;
int l1=0;
int l2=0;
int l3=0;
int F4=0;
int g1=0;
int g2=0;
int s1=0;
int s2=0;
int n1=0;
int n2=0;
int i1=0;
int i2=0;
for( int i = lmin-1; i <= lmax+1; i++ ){
    for( int j =0; j < 128; j++ )
    {
    if(thinned.getPixel(i,j)==0)
        pixelc++;
    }
}

for( int i = lmax+1; i <= 127; i++ ){
    for( int j =0; j < 128; j++ )
    {
    if(thinned.getPixel(j,i)==0)
        pixels++;

    }

}
int lowerv=0;
int upperv=0;
for(int i=0;i<total;i++)
{
if(tippositionx[i]<(minX+maxX)/2&&tippositiony[i]<(minY+maxY)/2)
    u1++;
if(tippositionx[i]>=(minX+maxX)/2&&tippositiony[i]<(minY+maxY)/2)
u2++;
if(tippositiony[i]<(minY+maxY)/2)

    lowerv++;
else
    upperv++;
}

int k=0;
for(int i=0;i<total;i++)
```

```
{
    if(tippositiony[i]>(maxY+minY)/2)
        xbottom++;
    else
        xtop++;
    if(tippositionx[i]>(minX+maxX)/2)
        xright++;
    else
        xleft++;
    if(tippositionx[i]>(2*minX+maxX)/3)
        k++;
}

        if(total>=4&&k>=2&&xbottom>=2&&xtop>=2&&pixels==0)
        {
            ActualChar[10][label]++;
            temp10=1;
        }



int c1=0;
for(int i=0;i<total;i++)
{

if(tippositionx[i]<(minX+maxX)/2)
    c1++;
}

        if(c1==0 && total<3)
        {

            ActualChar[2][label]++;

            temp10=1;
        }

        for(int i=0;i<total;i++)
        {
            if(tippositiony[i]<(minY+maxY)/2)
                y1++;
            else
                y2++;
            if(tippositiony[i]<(minX+maxX)/2)
                y3++;
            else
                y4++;

        }
            for( int i = lmax+1; i <= 127; i++ ){
            for( int j =0; j < 128; j++ )
            {
            if(thinned.getPixel(j,i)==0)
                pixelt++;
                    }
                }
```

```
        int pixelf=0;


        for( int i = lmax+3; i <= 127; i++ ){
            for( int j =0; j < 128; j++ )
            {
            if(thinned.getPixel(i,j)==0)
                pixelf++;
                    }
                }

int f3=0;
        for(int i=0;i<total;i++)
        {
            if(tippositiony[i]<(3*maxY+minY)/4)
                F3++;
            else
                F4++;
            if(tippositionx[i]>=(minX+2*maxX)/3)
                F1++;
            else
                F2++;
            if(tippositionx[i]<(2*minX+maxX)/3)
            f3++;
        }

        if(F3>=2&&F4>0&&F1!=0&&F2!=0&&f3>0&&pixelf==0)
        {
            ActualChar[5][label]++;
            temp=1;

        }
        int n3=0;
        for(int i=0;i<total;i++)
        {
            if(tippositiony[i]<(maxY+2*minY)/3)
                n1++;
            if(tippositionx[i]>(3*minX+maxX)/4)
                n2++;
            if(tippositionx[i]>(minX+maxX)/2)
           n3++;
        }

        int pixelq=0;
        for( int i = lmax+5; i <128; i++ ){
            for( int j =0; j < 128; j++ )
            {
            if(thinned.getPixel(i,j)==0)
                pixelq++;
                    }
            }


        if(n2>2&&n1>0&&n3>1&&total>2)
        {
            ActualChar[4][label]++;
```

```
    temp10=1;

}


if(F1==0)
{
    ActualChar[1][label]++;
temp10=1;

}

for(int i=0;i<total;i++)
{
    if(tippositiony[i]<(maxY+3*minY)/4)
   g1++;
    if(tippositionx[i]<(3*minX+maxX)/4)
    g2++;
}

int G1=0;
int G2=0;
for(int i=0;i<total;i++)
{
    if(tippositiony[i]<(maxY+3*minY)/4)
   G1++;
    if(tippositionx[i]<(3*minX+maxX)/4)
    G2++;
}
if(G2==0&&G1<2)
{
    ActualChar[6][label]++;
    temp10=1;
}

if(y1>=2&&y2==1&&pixelt<1)
{
    ActualChar[24][label]++;
    temp10=1;
}

y1=0;
y2=0;
int b2=0;
for(int i=0;i<total;i++)
{
    if(tippositionx[i]>=(minX+maxX)/2)
        b++;
}
if(b<1&&total<5)
{

    ActualChar[15][label]++;

    temp10=1;
}
```

```
if(u1==1&&u2==1&&r<2)
{
    ActualChar[20][label]++;


    temp10=1;
}

if(lowerv==2&&upperv<2&&u1!=0&&u2!=0)
{
    ActualChar[21][label]++;

    temp10=1;
}

int tex=0;
int tmid=(xlmin+xlmax)/2;

for( int ii = lmax+2; ii <= 127; ii++ ){
    if(thinned.getPixel(ii,tmid)==0)
        tex++;

}
i2=0;

int a[]=new int[128];
 for( int ii = 0; ii <= 127; ii++ )
 {
    a[ii]=0;
 }


     for( int ii = 0; ii < 127; ii++ ){


if((thinned.getPixel(((maxY+minY)/2)+1,ii)==1&&thinned.getPixel(((maxY+minY)/2)+1,ii+1)==
0))
        {

            i2++;
            a[ii]=1;

        }


    }

     if(i2==1)
     {
        int temp5=0;
        int ii3;
      for( int ii = 0; ii < 127; ii++ ){
```

```
if((thinned.getPixel((maxY+minY)/2+1,ii)==1&&thinned.getPixel((maxY+minY)/2+1,ii+1)==0))
                    {
                        ii3=ii;
                        while(ii3<127&&thinned.getPixel((maxY+minY)/2+1,ii3+1)==0)
                        {
                            ii3++;
                            temp5++;
                        }
                        if(temp5>4)
                        {
                            i2=2;

                        }

                    temp5=0;
                }
                  }
                }
int r7=0;
    int r8=0;
    int r9=0;
    int r10=0;
    int q2=0;
    for(int i=0;i<total;i++)
    {
        if(tippositiony[i]>(minY+2*maxY)/3)
            r7++;
        if(tippositiony[i]<(minY+maxY)/2)
            r8++;
        if(tippositionx[i]<(minX+maxX)/2)
            r9++;
        else
            r10++;
        if(tippositionx[i]<(2*minX+maxX)/3)
            q2++;

    }

    if(i2==2&&b<1)
    {
    ActualChar[3][label]++;
    temp10=1;
    }
    b=0;
    if(i2==1)
    {
        ActualChar[8][label]++;
        ActualChar[9][label]++;
        ActualChar[11][label]++;
        ActualChar[18][label]++;
        ActualChar[19][label]++;
        ActualChar[25][label]++;


        temp10=1;
```

```
    }
    if(i2>2)
    {
        ActualChar[13][label]++;
        temp10=1;
    }

    if(i2>=2&&q2==0)
    {
        ActualChar[16][label]++;
        temp10=1;
    }



    if(i2>1&&r8>1&&r7>1)
    {
        ActualChar[7][label]++;
        temp10=1;
    }

if(i2>1&&r7>1)
{

    ActualChar[0][label]++;
    ActualChar[12][label]++;
    temp10=1;

}

if(i2>1)
{
    ActualChar[17][label]++;
    temp10=1;
}

if(i2>1&&total>1&&r9!=0&&r10!=0&&r8!=0)
{
    ActualChar[22][label]++;
    temp10=1;
}

    if(xbottom>=2&&xtop==2&&xleft>=1&&xright>=1&&total>=4&&tex==0)
    {
        ActualChar[23][label]++;//x
        temp10=1;
    }

    if(i2>1 && total<3 && r7==0)
    {
        ActualChar[14][label]++;//o
        temp10=1;
    }
```

```
        r=0;
              if(temp10==0)
                  {

                     System.out.println(p);
                  }
              subclass[label]=subclass[label]+1;
        }


        }



is.close();
     Is.close();
     imageStream.close();
     labelStream.close();
     m_isLoaded = true;
     m_logger.info("Completed image loading");
   }



  public void Findmaxmin(int strX,int strY,int endX,int endY)
      {
      int Xmin=200,Ymin=200,Xmax=0,Ymax=0;
      int m=0,n=0,temp=0;
      int i=0,j=0;
      ImageInterface thinned2;
              try {
          thinned_image2 = m_thinner.apply(m_imageOne);
      } catch (ImageProcessingException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
      }

   for(j=0;j<thinned_image2.getHeight();j++)
      for(i=0;i<thinned_image2.getWidth();i++)
      {
       if((thinned_image2.getPixel(j,i)==0)&&(i<=Xmin))
          Xmin=i;

       if((thinned_image2.getPixel(j,i)==0)&&(i>=Xmax))
          Xmax=i;
      }
   for(j=0;j<m_imageOne.getHeight();j++)
      for(i=0;i<m_imageOne.getWidth();i++)
      {
       if((thinned_image2.getPixel(j,i)==0)&&(j<=Ymin))
           Ymin=j;

       if((thinned_image2.getPixel(j,i)==0)&&(j>=Ymax))
           Ymax=j;
```

```
    }

 if(Xmin==200)
    Xmin=0;
 if(Ymin==200)
    Ymin=0;



 maxY=Ymax;
 minY=Ymin;
 minX=Xmin;
 maxX=Xmax;
}
public  int pixelCount(int strx,int stry,int endx,int endy) {
    int numberOfPixels = 0;

    for( int i = stry; i <= endy; i++ ){
        for( int j =strx; j < endx; j++ ){

if( m_imageOne.getPixel(i,j)== 0 )
    numberOfPixels++;
        }
            }

    return numberOfPixels;
}


public void ExtremeEdges(int strX,int strY,int endX,int endY)
{

    EXmin=200;EXmax=0;EYmin=200;EYmax=0;

    try {
        thin = m_thinner.apply(m_imageOne);
    } catch (ImageProcessingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    //Finding the left extreme edge
     for(int i=strY;i<endY;i++)
        for(int j=strX;j<endX;j++)
        {
          if((thin.getPixel(j,i)==0)&&(j<=EXmin))
              EXmin=j;

          if((thin.getPixel(j,i)==0)&&(j>=EXmax))
              EXmax=j;
        }

      for(int j=strY;j<endY;j++)
       for(int i=strX;i<endX;i++)
        {
          if((thin.getPixel(i,j)==0)&&(j<=EYmin))
```

```
                    EYmin=j;

                if((thin.getPixel(i,j)==0)&&(EYmin>=0))
                    EYmax=j;
                }

            if(EXmin==200)
                EXmin=0;

            if(EYmin==200)
                EYmin=0;

            minX=EXmin;
            maxX=EXmax;
            minY=EYmin;
            maxY=EYmax;
            System.out.println("EXmin :"+EXmin+"
EXmax :"+EXmax+"ymin"+EYmin+"ymax"+EYmax);
    }


    public void  NumberOfTips2(int [][]imagearray)//to remove the tips of the unthinned image
    {  int repeat=0;
        int tipcnt=0;
        int r=0;
        int[] RotX=new int[8];
        int[] RotY=new int[8];

        RotY[0] = -1;
        RotY[1] = -1;
        RotY[2] = -1;
        RotY[3] = 0;
        RotY[4] = 1;
        RotY[5] = 1;
        RotY[6] = 1;
        RotY[7] = 0;

        RotX[0] = -1;
        RotX[1] = 0;
        RotX[2] = 1;
        RotX[3] = 1;
        RotX[4] = 1;
        RotX[5] = 0;
        RotX[6] = -1;
        RotX[7] = -1;
        int []x=new int [20];
        int []y=new int [20];
        int k=0;
        while( repeat==0)
        {
        for( int i=1;i<127;i++)
        {
         for( int j=1;j<127;j++)
         {
             tipcnt=0;
             if(imagearray[j][i]==0)
```

```
        {
            for(int p=0;p<8;p++)
            {
                if((imagearray[j+RotX[p]][i+RotY[p]]==0))
                {

                    tipcnt++;
                }
            }
        }
            if(tipcnt==1)
            {
                x[k]=i;
                y[k]=j;
                k++;
                tipcnt=0;
            }

            }
    }
    if(k==0)
    {
    repeat=1;
    }
    for(int q=0;q<k;q++)
     {
        imagearray[y[q]][x[q]]=1;

 }
    k=0;
     }
}


public int NumberOfTips(int strX,int strY,int endX,int endY)
{
    int tips=0;
    int tipcnt=0;
    int r=0;
    int[] RotX=new int[8];
    int[] RotY=new int[8];

    RotY[0] = -1;
    RotY[1] = -1;
    RotY[2] = -1;
    RotY[3] = 0;
    RotY[4] = 1;
    RotY[5] = 1;
    RotY[6] = 1;
    RotY[7] = 0;

    RotX[0] = -1;
    RotX[1] = 0;
    RotX[2] = 1;
    RotX[3] = 1;
```

```
            RotX[4] = 1;
            RotX[5] = 0;
            RotX[6] = -1;
            RotX[7] = -1;


            try {
                thinned_image = m_thinner.apply(m_imageOne);
            } catch (ImageProcessingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }



            for( int i=strY+1;i<endY-1;i++)
             for( int j=strX+1;j<endX-1;j++)
             {
                tips=0;
                if(thinned_image.getPixel(j,i)==0)
                    for(int p=0;p<8;p++)
                    {
                        if((thinned_image.getPixel(j+RotX[p],i+RotY[p])==0))
                        {

                            tips++;
                        }
                    }

                    if(tips==1)
                    {

                        tippositionx[r]=j;
                    tippositiony[r]=i;
                        tipcnt++;
                        tips=0;
                        r++;
                    }
             }


            for( int i=0;i<tipcnt;i++)
            System.out.println("the tip positionx:"+tippositionx[i]+"tippositiony:"+tippositiony[i]);


            return tipcnt;
        }

        public ImageInterface RegUsingEdgeProcess() throws ImageProcessingException,
    IOException
        {
            int grpA=0;
            int grpB=1;
            int []sum=new int[26];
           char ch[]={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
            Runtime r=Runtime.getRuntime();
```

```
  double [][][][]P=new double[4][26][40][20000];

  //double [][][]ResVec=new double [26][26][parts*(features)];
  setMaxImages(2500);
  try {
       restoreData(P);//"data/MNIST");
  } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
  }

            for(int i=0;i<=25;i++)

  System.out.println("number of "+" " +ch[i]+ " " +subclass[label]);

for(int i=0;i<26;i++)
{
System.out.println("classification for the character    "+ch[i]);
  for(int j=0;j<26;j++)
  {
       System.out.println(" Number of "+ch[j]+"'s :"+ActualChar[i][j]);
}
}



for (int i=0;i<26;i++)
{
  for(int j=0;j<26;j++)
  {
  outputchar[j][i]=ActualChar[i][j];
  }
}

for (int i=0;i<26;i++)
{
  for(int j=0;j<26;j++)
  {
       sum[i]=sum[i]+outputchar[i][j];
  }
  }

for(int i=0;i<26;i++)
{
System.out.println("classification for the character    "+ch[i]);
  for(int j=0;j<26;j++)
  {
       System.out.println(" Number of "+ch[j]+"'s :"+outputchar[i][j]);
}
   System.out.println("total characters    "+ch[i]+"recognized as other characters"+sum[i]);
}

System.out.println("Total characters analyzed : "+images);
```

```
            r.freeMemory();

                return shadow3_image;
        }


}
```

VITA

Graduate College
University of Nevada, Las Vegas

Koushik Reddy Damera

Home Address:
4210 Cottage Circle Apt # 01
Las Vegas, NV, 89119

Degree:
Bachelor of Engineering, 2004
Osmania University, Hyderabad, India.

Thesis Title: Hierarchical Approach for Character Recognition.

Thesis Examination Committee:
Chair Person, Dr. Evangelos A. Yfantis, Ph.D.
Committee Member, Dr. Ajoy K Datta, Ph.D.
Committee Member, Dr. Yahoowan Kim, Ph.D.
Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.