Chapter 8

Feature Extraction and Recognition of Characters using Vector Contour

8.1 Introduction

In the previous chapter, a new and effective approach to extract the features of the characters has been described. This approach is capable for not only to recognize the characters but any object, having any shape. In this chapter we present a new approach to recognize the characters of any types, just by extracting their features, based on vector contour.

The process involved in our approach is very similar to existing approaches as shown in figure 8.1:

1. Handwritten characters
2. Optical scanner for Digitalization
3. Isolation of characters
4. Preprocessing for Normalization and Thinning
5. Feature Detector (Matching)

* The entire chapter in the form of papers has been published in two parts: "ARPN Journal of Systems and Software", VOL. 2, NO.7, pp. 228-235, July 2012 and "International Journal of Artificial Intelligence & Applications (IJAIA)", Vol.4, No.3, pp.23-38, May 2013.

6. Identity of characters (Recognition)

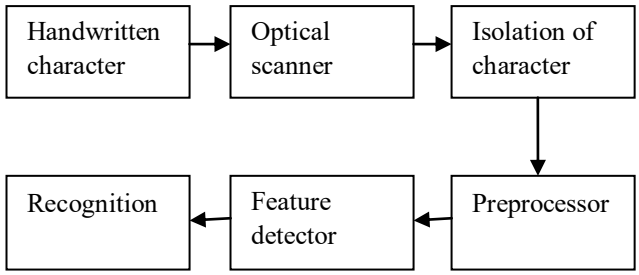

Figure 8.1: General Block diagram of handwritten character recognition

In pre-processing phase of characters, the character is bounded for normalization for standard size and thereafter thinning of character for noise removal or skeletonizing is done. The following steps are involved in our Preprocessing phase-

1. Extraction of a character in the given word.
2. Position normalization of character
3. Normalization of character
4. Thinning of normalized character

The preprocessing phase can be done using the standard techniques available.

8.2 Related concepts

8.2.1 Character Normalization

Normalization is done to make the size of all extracted character bitmaps equal. In order to match the extracted isolated character, it is important that all patterns should have the same size. So, size normalization is required. Size normalization is the most efficient pre-processing technique. However, the use of pre-processing techniques depends on many factors such as quality and shape of the data and the recognition process employed.

There are various techniques available for normalization which can be referred from the test books or research papers. For the sake of completeness we try to explain through figure 8.2. In this method, every input bitmap P, of dimension

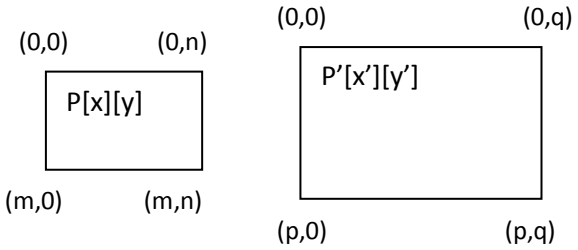

Figure 8.2: Normalization

$'m \times n'$, is transformed into a normalized bitmap P', of dimension $'p \times q'$. The geometric transformations generally modify the spatial relationships between pixels in an image. They are also termed as rubber sheet transformations.

8.2.2 Thinning

Thinning is the process to extract and apply additional constraints on the pixel elements that are to be preserved so that a linear structure of the input image will be recaptured without destroying its connectivity. Thinning plays a very important role in the pre-processing stage of pattern recognition. This is due to the fact that:

- It preserves essential structural information of an image.
- It reduces the space to store topological as well as shape information of an image.
- It reduces the complexity of analyzing the image.

In the context of two - dimensional binary image processing, thinning is considered as an recursive process of removing points or layers of outline from a binary image until all the lines or curves becomes single pixel wide. The reduced pattern is called the skeleton. A good thinning algorithm must preserve the topology as well as the shape of the original image in the skeleton.

Many thinning algorithms (or modifications of existing ones) have been proposed in recent years, and a comprehensive survey of these methods is contained in Lam *et al.* [194]. However, to name a few, Naccache and Singhal [205] made a study of fourteen thinning algorithms based on iteration erosion of boundary. They

proposed the safe point thinning algorithm (SPTA). Saha *et al.* [206] proposed an improvement on the algorithm by suggesting a rotational invariant single scan boundary removal thinning algorithm (SBRTA). Lu and Wang [207] suggested an improvement on this. Lam and Suen [208] evaluated another ten thinning algorithms.

8.3 Feature Extraction and character identification

A novice method, termed as character divider approach for feature extraction of characters followed by the preprocessing steps is presented. There are 7 phases, performed in our approach and is illustrated in the figure 8.3.

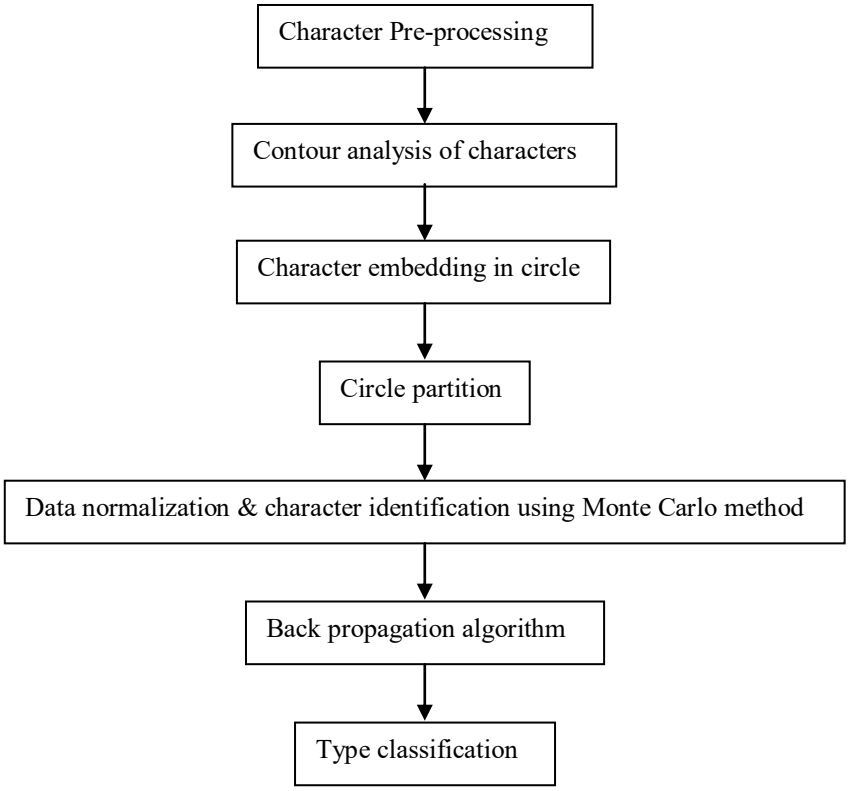

Figure 8.3: Phases for character identification

8.3.1 Phase1: Character Preprocessing

In this Phase the preprocessing of the character (as discussed in section 8.1) can be done in three phases, given in figure 8.4.

We perform 3 steps for character isolation, illustrated below:

Step1: Extraction of Lines- The text image is scanned from top to bottom and from left to right to extract number of lines in the written text. The algorithm is given as Algorthim 1.

Step2: Extraction of Character- Within each line, top-left and bottom-right co-ordinates of each character is obtained using the algorithm, given as Algorthim 2.

Step3: Position normalization of characters- For each character array obtained from above two algorithm, redundant rows of '0's are removed to get the actual character array, which fits into a bounded box.

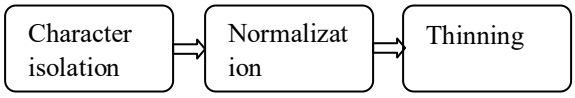

Figure 8.4: Phases of character preprocessing

Data: int inpu[rows][cols];
//input image binary image array
int line_count;
Result: Line extracted
initialization;
while *all lines got scanned* **do**
    read current;
    if *scan[i][j] == 1* **then**
        setline[line_count].start.x=1;
        goto next row of array and look for '1' in that row;
    end
    if *'1' found* **then**
        goto next row and so on till a row (kth row) of all '0's is encounterd;
        setline[line_count].end.x = k-1;
        //So, first line is obtained.;
        line_count++;
    end
end

Algorithm 1: Algorithm for Line Extraction

Following above three steps, all the characters of different sizes, in separate 2D binary array. So we need normalization as given in section 8.2.1 and illustrated in detail as under: In the two quadrilateral regions, given in figure 8.2; the vertices correspond to the tie points. The geometrical transformation process within the regions is modeled by a pair of bilinear equations so that

$$x' = c_1x + c_2y + c_3xy + c_4$$
$$y' = c_5x + c_6y + c_7xy + c_8$$

These equations can easily be resolved for 8 coefficients $c_i, i = 1, \ldots, 8$. Once the coefficients are known, they constitute the model used to transform all pixels within the quadrilateral region characterized by tie points used to obtain the coefficients.

Simplifying the above equations, we get,

$c_1 = \frac{p}{m}; c_6 = \frac{q}{n}; c_2, c_3, c_4, c_5, c_7, c_8 = 0$

Therefore,

$$x' = (\frac{p}{m})x$$
$$y' = (\frac{q}{n})y$$

If we try to normalize the thinned character skeleton, the resulting skeleton is not found to be connected. So, the un-thinned character is normalized and later thinned. Reverse mapping is done for normalization i.e.

1. For each pixel position $P'[r][c]$, the equivalent pixel position $P[i][j]$ is found using above relation (many pixel positions of Q may map to the same pixel position of P)

2. The value (0 or 1) at $P'[r][c]$ pixel is set to that at the location $P[i][j]$

```
Data: char scanchar[j];
Result: Character extracted
while getline do
    for(int i<0; i<j; i++){
    if scanchar[i]=='1' then
        char[i].top_left.x=line[line_count].start.x;
        char[i].top_left.y=that col no.;
    end
    if scanchar[i]=='0' then
        char[i].bottom_right.x=line[line_count].end.x;
        char[i].bottom_right.y= that col no.;
    end
end
//character gets extracted
```

Algorithm 2: Character extraction algorithm

After normalization, we do the thinning as per the Algorithm 3.

Data: char scanbitmap[j];

Result: Thinning algorithm

while *no more pixels can be deleted* **do**

Scan binary bitmap in 4 directions: up-down, down-up, left-right, right-left.;

In each pass, pixels which are not simple and have more than one eight neighbors (i.e. they are not end points) are deleted;

Conditions for deletion of a point from a given side of the matrix should be checked before any other points are deleted. On each row points are marked for deletion, but are not deleted until the points on the following row have been marked.;

end

//The resulting bitmap constitutes a skeleton of the original pattern i.e. a one-pixel wide pattern consisting of a subset of the original black pixels.

Algorithm 3: Thinning algorithm

8.3.2 Phase2: Contour analysis of characters for its pre-classification

In this phase we do the contour analysis for each character starting from A to Z. The contour contains the necessary information on the object shape. We define contour as a boundary of an object, a population of pixels, separating object from a background. We have encoded contour by the sequence consisting of complex numbers. On a contour, the starting point is fixed. Starting point should be the centroid of the pixel. Then, the contour is scanned (is admissible - clockwise or anticlockwise depending on the continuity of the connectivity of the adjacent pixels), and each vector of offset is noted by a complex number $a + ib$, where a is the point offset on x axis, and b is the offset on y axis. Offset is noted concerning the previous point.

To illustrate our approach for defining the vector contour of a character, we create the vector contour of 'B' as given in figure 8.5. It is to be noted that from the starting point there is no adjacent pixel in clockwise direction, hence anticlockwise direction has been considered for deriving the vector contour.

So Vector Contour (VC) is the set of vectors, known as elementary vectors(EV) which define the contour of the character. Hence for character 'B', the VC defines as:

$$VC_B = \{i, i, i, i, i, i, 1, 1-i, -i, -1-i, -1, 1, 1-i, -i, -1-i, -1\}$$

We have pre-classified all the capital letters based on their vector contours. Similarly we can compute the VC for all characters as shown for 'B' in figure 8.5(P-118):

$$VC_A = \{-1-i, -i, 1-i, 1+i, i, -1+i\}$$

$$VC_B = \{i, i, i, i, i, i, 1, 1-i, -i, -1-i, -1, 1, 1-i, -i, -1-i, -1\}$$

$$VC_C = \{-1, -1-i, -1-i, 1-i, 1-i, 1\}$$

$$VC_D = \{-i, -i, -i, 1-i, 1+i, 1+i, i, -1+i, -1+i, -1-i\}$$

$$VC_E = \{-1, -1, i, i, 1, -1, i, i, 1, 1\}$$

$$VC_F = \{i, i, 1, -1, i, i, 1, 1\}$$

$$VC_G = \{-1, -1-i, -1-i, 1-i, 1-i, 1, i, -1\}$$

$$VC_H = \{-i, -i, -i, -i, i, i, 1, 1, i, i, -i, -i, -i, -i\}$$

$$VC_I = \{-1-i, -i, 1-i, 1-i, -i, -1-i, -1+i, i, 1+i, 1+i, i, -1+i\}$$

$$VC_J = \{1, 1, -1, -i, -i, -i, -i, -i-1, -1, -1+i\}$$

$$VC_K = \{-i, -i, -i, -i, i, i, 1+i, 1+i, -1-i, -1-i, 1-i, 1-i\}$$

$$VC_L = \{-i, -i, -i, -i, 1, 1\}$$

$$VC_M = \{i, i, i, i, 1-i, 1-i, 1+i, 1+i, -i, -i, -i, -i\}$$

$$VC_N = \{i, i, i, i, 1-i, 1-i, 1-i, 1-i, -i, -i, -i, -i\}$$

$$VC_O = \{-1-i, -1-i, -i, 1-i, 1-i, 1, 1+i, 1+i, i, i, -1+i, -1+i, -1\}$$

$$VC_P = \{i, i, i, i, 1+i, 1-i, -1-i, -1+i\}$$

$$VC_Q = \{-1-i, -1-i, -i, 1-i, 1-i, 1, 1+i, -1+i, 1-i, 1-i, 1+i, i, i, -1+i, -1+i, -1\}$$

$$VC_R = \{i, i, i, i, 1+i, 1-i, -1-i, -1+i, 1-i, 1-i, 1-i\}$$

$$VC_S = \{-1+i, -1+i, -1, -1-i, -1-i, 1-i, 1-i, 1-i, -1-i, -1-i, -1, -1+i, -1+i\}$$

$$VC_T = \{1, 1, -1, -i, -i, -i, -i\}$$

$$VC_U = \{-i, -i, -i, 1-i, 1, 1+i, i, i, i\}$$

$$VC_V = \{1-i, 1-i, 1-i, 1+i, 1+i, 1+i, 1+i\}$$

$$VC_W = \{1-i, 1-i, 1-i, 1+i, 1+i, 1+i, 1+i, 1-i, 1-i, 1-i, 1+i, 1+i, 1+i, 1+i\}$$

$$VC_X = \{1-i, 1-i, 1-i, 1-i, -1+i, -1+i, -1-i, -1-i, 1+i, 1+i, 1+i, 1+i\}$$

$$VC_Y = \{1-i, 1-i, 1+i, 1+i, -1-i, -1-i, -1-i, -1-i\}$$

$$VC_Z = \{1, 1, 1, -1-i, -1-i, -1-i, 1, 1, 1\}$$

Table 8.1: VC of characters & its VE

VC character	**No. of Vector elements**	**No. of bits required**
A,C,L	6	3
B,Q	16	4
D,E,J	10	4
F,G,P,Y	8	3
H,W	14	4
I,K,M,N,X	12	4
O,S	13	4
R	11	4
T,V	7	3
U,Z	9	4

We define the VC for characters as the set of vector contour of all characters.

$$VC = \{VC_A, VC_B, VC_C, VC_D, VC_E, VC_F, VC_G, VC_H, VC_I, VC_J, VC_K, VC_L, VC_M, VC_N,$$
$$VC_O, VC_P, VC_Q, VC_R, VC_S, VC_T, VC_U, VC_W, VC_X, VC_Y, VC_Z\}$$

which is a two dimensional array. Table 8.1 illustrates the number of vector elements in each of the VC set and hence the number of bits to represent them by knowing that 1 bit is sufficient to represent 2^1 states/values and 2 bits are sufficient to represent 2^2 states/values and so on...

8.3.3 Phase 3: Character embedding in circle

In this phase we circumscribe the character with a circle to create the boundary to be analyzed for its (character) identification, as shown in figure 8.6.

8.3.4 Phase 4: Circle partition

In the previous phase, the centroid of each character pixel (black pixel 1's) has already been computed. Subsequently, in this phase, division of circumscribed circle into eight parts (arcs) with centroid is carried out so that each partition can take the range of 45 degrees, which has been assumed to be optimized value

Table 8.2: Partition number and its range

Partition Number	**Partition Range R (in degree)**
1	$0 < R \leq 45$
2	$45 < R \leq 90$
3	$90 < R \leq 135$
4	$135 < R \leq 180$
5	$180 < R \leq 225$
6	$225 < R \leq 270$
7	$270 < R \leq 315$
8	$315 < R \leq 360$

with respect to the computation and information integrity. Table 8.2 describes the range of degrees for each partition in circle.

8.3.5 Phase5: Data Normalization & character identification using Monte Carlo Method

We use Monte Carlo method as described in [93] for data normalization in which the following steps are performed: In each partition of the circumscribed circle, count the number of pixel's (1's), say np_i , where i is the partition number of circle. So range of i is from 1 to 8 . Count the total number of pixels present in the character array, say np_t.

Do normalization by performing $\frac{np_i}{np_t}$

We have shown the enclosed character 'A' with its partition in figure 8.6 (P-119).

In each partition, angle of each pixel is found which varies between particular partition's minimum range and maximum range. The summation of all angles of all the pixels of each partition is done and then normalization of this partitioned data vector is carried out. Then the centroid of the character i.e. the x and y co-ordinates is taken and normalized and is also considered as data for identification of characters. In all we got 10 partition data. This data is collected for all input characters set written by different individuals. It uses the same pre-classification as described in Phase 2, using vector contour.

8.3.6 Phase 6: Back propagation Algorithm

We have preclassified the characters using VC in phase 2. Then in phase 5, we extracted the data for each partition for the given character. In this phase, this partition data is fed as input to back propagation algorithm. Back propagation algorithm is given in Algorithm 4.

while *all [error(k)]*$\leq$ *0* **do**

for(each training pattern){

forward pass: calculate the output of 3 layers;

First layer://fans out the i/p to next layer;

Output[1,i]=input[1,i];

Second layer:;

input[2,j]=weight1[i,j]×o/p[1,i];

output[2,j]=1.0/(1.0+exp(-input[2,j]);

Third layer:;

input[3,k]=weight2(j,k)×output[2,j];

compute error:;

error[k]=output[3,k]-target[k];

Backward pass the error:;

δw2[j,k]=η×output[3,k]×(1-

output[3,k])×error[k]×output[2,j]+α × δw2[j,k];

output[3,k]×(1-output[3,k])×error[k]×weight2[j,k]+ α × δw1[i,j];

weight1[i,j]=weight1[i,j]- δw1[i,j];

weight2[j,i]=weight2[j,k]- δw2[j,k];

}

end

Algorithm 4: Back propagation algorithm

Networks are trained according to following Back propagation neural network classification which is based on the VC, given in table 8.3.

Reason for taking 26 input nodes in each topology is because from the character 26 circle data are extracted i.e. 26 features of characters are extracted. The

Table 8.3: Topology for EV

Topology No.	**No. of Input nodes**	**No. of hidden nodes**	**No. of output nodes**	**Character**
1	26	20	3	A, C, L, F, G, P, Y, T, V
2	26	20	4	B, Q, D, E, J, H, W, I, K, M, N, X, O, S, R, U, Z

number of output nodes has been derived from the number of bits required as shown in table 8.1 (P-110). There is no clear rule for the 'best' number of hidden nodes, some authors propose to assign the number of hidden nodes by the following formula:

$$No.of hiddennodes = \frac{no.of inputnodes + no.of outputnodes}{2}$$

The experienced mathematical researches can assign the number of hidden layers using Fisher Information Matrix [209]. Characters A,C,L,F,G,P,Y,T,V are trained with 26-20-3 configuration and rest of the characters are trained with 26-20-4 configuration.

Initially these networks are set up with weight generated from random number generator which may also affect the performance of the resulting Multi-layer Feed-Forward (MLF) ANN. These weights are adjusted during training to desired output as per the following procedure:

1. The input and output of each node (j) in the hidden and output layers are computed.

   $$I_j = \sum_i w_{ij} o_i + b_j$$

   $j = 1, 2, \ldots, 8$ (no.of training data).

   w_{ij}-weight of the connection from node i in the previous layer to node j.

   o_i-Output of node i in the previous layer.

b_j-bias of node j.

$$o_j = f(I_j) = \frac{1}{1 + e^{-I_j}}$$

which is non linear and differentiable.

2. Search for a set of weights that fits the training data such that the mean squared error (MSE) can be minimized, using gradient descent method. Error of node j is given by

$$Err_j = \begin{cases} o_j(1 - o_j)(T_j - o_j) & \text{j is an output node} \\ o_j(1 - o_j)\sum_k e_k.w_{jk} & \text{j is a hidden node} \end{cases} \tag{8.1}$$

o_j-the output of node j

T_j-the true target value of node j

e_k-the error of node j in the next layer

w_{jk}-the weight of connection from node j to node k

Change in weight $\Delta w_{ij} = (I)e_j o_i$

Change in bias $\Delta b_j = (I)e_j$

l- learning rate (a rule of thumb):$l = \frac{1}{t}$, t is the number of iterations so far

$$w_{ij} = w_{ij} + \Delta w_{ij}$$
$$b_j = b_j + \Delta b_j$$

3. Stop the process till all Δw_{ij} are below some specified threshold

So, in total 8 training files which consist of extracted partition data of character are trained and subsequently 8 weight files are generated.

8.3.7 Phase 7: Type classification

From table 8.3 (P-113), it can be seen that there are some elementary vectors in set VC (Phase 2) which do have either horizontal or vertical or both symmetry.

VC_0, VC_1, VC_3, VC_8.

So in VC, 3 sub groups lie:

1. Horizontal symmetry based sub group number 1: (VC_C, VC_E, VC_I)
2. No symmetry based sub group number 2: $(VC_F, VC_G, VC_J, VC_L, VC_N, VC_S), VC_Z)$
3. Vertical symmetry based sub group number 3: (VC_M, VC_U, VC_V, VC_W)

Since within the set of VC, characters can be placed in 3 subgroups as discussed above. So there is need for two stage back propagation neural classifiers, which is described below:

A neural network having d input nodes, c output nodes can be considered as a classifier to assign a given sample with d feature to one of 3 predefined sub group numbers.

During recognition of unknown character it goes through 2 stage process. During its first stage, the type classifier gets activated which indicate whether the character is from horizontal or no symmetry or vertical symmetry subgroup. Then according to the identification of subgroup from first stage, in second stage particular type recognition network (type 1 or type 2 or type 3) gets activated which finally gives true identity of character.

8.4 Pros and Cons of two stage Back propagation neural classifier

There are two main advantages of using 2-stage Back propagation neural classifier:

1. A single problem can be decomposed into many small sub-problems, which are easier to manage.
2. The output node of the ANN can be reduced for efficiency purpose.

If we look into the disadvantages, we could found only one that is if the first classifier fails to interpret correctly, then the wrong output of the first classifier will become the input of the second classifier, which will give the wrong prediction. But that can always be minimized by making the use of efficient training algorithm to train the hidden layer of ANN.

Table 8.4: Data from different persons

Subgroup number	1	2	3
Training sample number	100	147	123

8.5 Experimental Results

8.5.1 Sample Text Files

Some test documents containing character sets written by different individuals in their handwriting was collected and scanned using scanner to generate corresponding tiff files and corresponding to which binary files were formed and processed. First lines were separated out from text followed by separation of each character which were then enclosed in a bounding box, then binaried, normalized and thinned using a thinning algorithm. Some test documents containing phrases on different lines were tested during experiments. A sample text file is shown in figure 8.7 (P-119).We collected character set in different handwritings from A-Z in a file.

8.5.2 Pre-processing of characters

The characters are pre-processed for identification. Figure 8.8 (P-119) shows the result of 'H' extraction and figure 8.9 (P-120) shows feature extraction (after thinning), from sample text file1 (figure 8.7).

8.5.3 Normalized circle data of file

Normalized circle data of file for 'H' is only shown.

0.086957 0.152174 0.196552 0.326078 0.108696 0.134351 0.134509 0.156900 0.189036 0.118818 0.153421 0.247432 0.339870 0.077045 0.066120

Several experiments were carried out to demonstrate the performance of this data. For training and testing total number of samples taken from different persons is shown in table 8.4.

Total number of samples for training = 370

Recognition rate of this approach found is 76%

8.6 Limitations

There are some limitations to our approach for character identification which are give as under:

1. Sometimes the person can write the characters which are not purely separated due to hurry-burry or the individual's style of writing. In that case our Vector contour will not be appropriate and our system would have to train for large data sets and for longer duration to predict the given character correctly.

2. We have partitioned the circumscribed circle into 8 parts. As the number of partitions increases, the prediction would be more accurate but the complexity will be increased and the designer must have to analyze the proper balancing in between the number of partitions and its complexity.

3. We have used MLF ANN, for which the training time can be very lengthy.

There is a need of rather a relatively large dataset for training to get accurate predictions.

8.7 Conclusion

We introduced a novel set of features that is well-suited for representing handwritten characters. The features are derived from the Vector Contour (VC) set and symmetrical nature of the characters. For the sake of simplicity during the VC analysis we assumed the value of a and b in $a+ib$ as unity though actual value have to be trained by collecting large data set. The feed forward neural network was trained for VC and symmetrical nature identification which collaboratively is the very unique feature for character, even VC itself is the most unique feature, if the values of a and b are given correctly. Our character recognition system can also be used for numeral recognition. We have introduced a mechanism for handwritten character recognition and accordingly all the related algorithms.

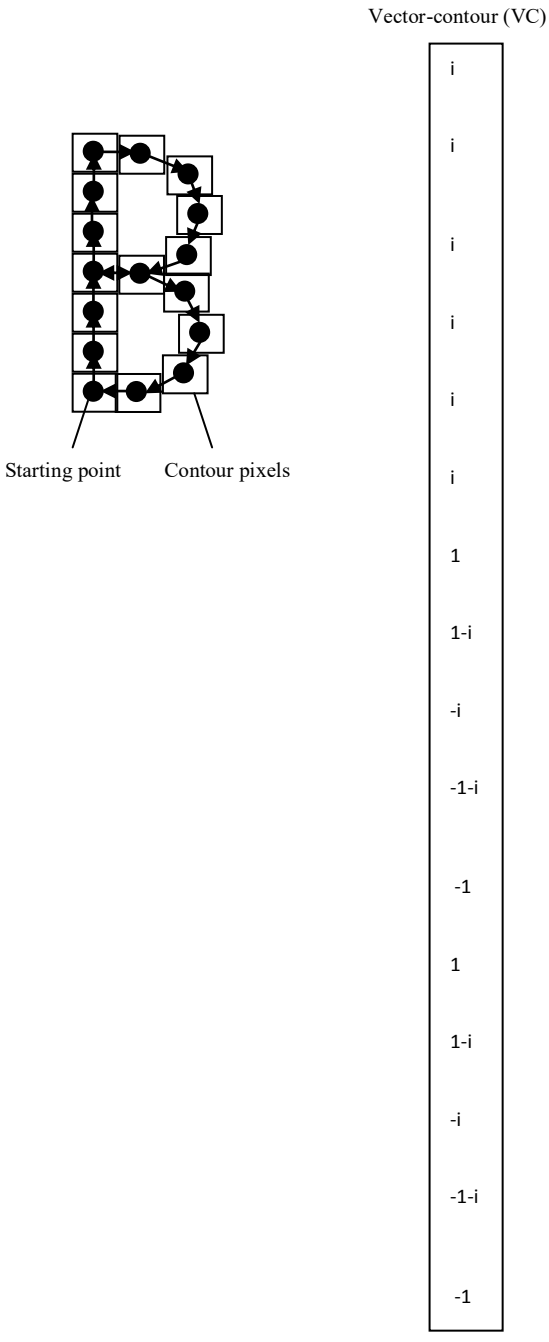

Figure 8.5: Vector Contour of B

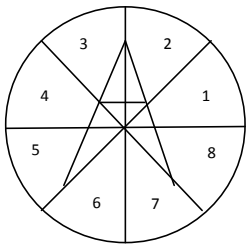

Figure 8.6: Circle partition with character 'A'

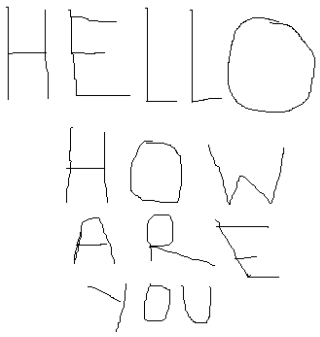

Figure 8.7: Sample Text File1

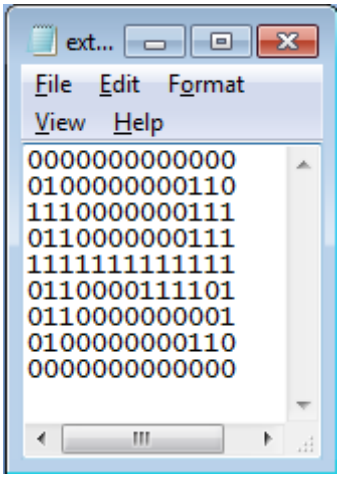

Figure 8.8: 'H' extracted from input text file

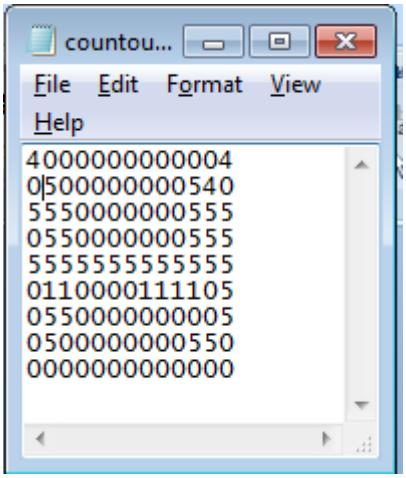

Figure 8.9: Feature extraction (after thinning)