3 Handwriting Recognition

3.1 Signature Search and Verification using Human Perspective

3.1.1 Introduction

Off-line Signature search and verification is researched by computer scientists for many years with different approaches [251,252] using global features, structural features [48], statistical features, and density features.

To what level should one compare, to say that signatures are same or not is the basic problem. Two signatures of the same person, if seen minutely, show a lot of variations but are still to be considered same. Forged and genuine signature of a person look same while viewing the overall structure. Signatures of different persons may vary globally or at minute levels. In signature recognition, style of writing is to be considered and contents ignored while in case of character [22,253] recognition contents is to be considered and style of writing is ignored. With such constraints, signature search and verification remains a challenging research area.

Signature recognition deals with searching of signature database to associate identity of a given signature while signature verification separates out a genuine signature from a forged signature. The objective of the signature search is to provide an efficient and reliable technique for identifying the correctly entry in a database of signatures.

Forgeries can be considered as 'random', simple and skilled [254]. Random forgery uses signature of other persons as forged signatures. Simple forgery makes no attempt to trace a genuine signature while skilled forgery tries to imitate as closely as possible the dynamic and static information of a genuine signature.

This research proposes Signature search and verification using a different approach
[256], presuming it to be as done by human beings. Humans while searching and verifying signature or any object as such, have in their mind the purpose of doing so. This purpose and the domain knowledge helps him to search and verify easily and quickly. The purpose decides the level to which comparison should be made to be satisfied with search and verification. For example the amount on the cheque influences cashier’s view of verifying signature. Higher the amount, more precise is the checking.

Human looks initially at the overall structure, a coarse view, of an object. If he is not satisfied with the comparison, goes gradually into detail, a finer view. That is he gets first less information which is global in nature, followed by more information which is more local in nature to the extend that he gets desired results. To exactly replica the human behavior, a signature sample can be scanned at different resolutions going from low to high gradually. But this procedure will be time consuming, hence scanning with higher resolution is done once and the amount of information needed, is extracted for processing.

The variation in the signature specimens of the same person and the noise due to quality of paper, ink and scanning of signature sample, restricts the amount of details one can use to distinguish signature samples. On the other hand, as said earlier, there do exist global structural difference between signature samples of different people which help to classify them into sub-classes which have less number of signature samples. These reduced number of signature samples can be separated using more detailed view.

3.1.2 The proposed models

The models are 'Human' model, 'Machine' model and 'Hybrid' model. These models have five levels, starting with a global view going down to a detailed view of signature sample. The first level is of width-to-height ratio of signature sample. It is followed by three different view levels of the structure of the blots formed by signature samples. The final level
is of multiple neural networks. The term 'Blot' is used here to define a dilated image as if ink diffuses on a piece of paper. The three models differ in the method of connecting the five levels. Figures 3, 4 and 5 show 'Human', 'Machine' and 'Hybrid' model respectively.

In the 'Human' model, if at any level a single matching signature sample is found, the signature sample is accepted. In the 'Machine' model, it is compulsory for signature sample to go through all levels for matching, hence its name. The 'Hybrid' model is combination of 'Human' and 'machine' models. The first four levels behave like 'Human' model while the last level is compulsory as in 'Machine' model.

At any level, if more than one matching signature samples are found they are send to next level. In case at any level no matching signature sample is found, the model indicates that it cannot classify signature sample.

There are two phases for a model. First is the Knowledge generation phase in which signature samples in training set are analyzed to generate knowledge base for each level, deciding threshold parameters and selecting training patterns for Neural networks using a clustering algorithm. Second is the Signature Search and Verification phase which is used to search and verify whether the unknown signature sample is genuine or not.

WHR knowledge base stores the mean of width-to-height ratio of signature samples for each person. Knowledge Base of Blot-9, Blot-6 and Blot-3 store mean of corresponding blot vectors of signature samples and the threshold parameters. Neural network knowledge base stores trained neural networks and a file containing information about association of a person's signature with different neural networks.
In put

Output

(zero, one, more indicate numbers of matching signatures)

Fig. 3 : Human Model
Fig. 4: Machine Model

(zero, one, more indicate numbers of matching signatures)
(zero, one, more indicate numbers of matching signatures)

**Fig. 5**: Hybrid Model
3.1.3 Signature normalization

The black and white image of Signatures are normalized to a grid of 16 × 12 cells. Aspect ratio of 4:3 is selected as in most of the signature samples, it is found that width is larger than height. For each cell, the gray level is calculated as in equation (15):

\[
GrayLevel_k = \frac{\sum_{i=1}^{m_k} \sum_{j=1}^{n_k} I_{i,j}}{m_k \cdot n_k} \cdot (L-1)
\]  

(15)

where, Gray Level$_k$ is the gray level of the kth cell, $m_k$ and $n_k$ are number of pixel rows and columns in the kth cell respectively, $I_{i,j}$ is the black/white intensity of a pixel in a cell and $L$ is the number of gray shades.

Thus a normalized signature sample is stored as vector with 192 elements with each element having a value between 0 to L-1.

3.1.4 WHR level

The rectangular enclosure around a signature sample is used to get its width to height ratio. For all signature samples in training set, the mean of width to height ratio of signature samples for each person are stored in WHR knowledge base. Figure 6 show the WHR for 60 persons along with their deviation by line segments.

For an unknown signature sample, its width to height ratio (WHR) is calculated and is used for finding matching signature samples. It is found on analysis that higher the magnitude of a signature sample, higher is the deviation in WHR, hence instead of fixed range, dynamic range is taken to form cluster of matching signature samples using range_factor 'f' as given in (16), (17) and (18):

\[
R_{\text{start}} = \text{WHR} \cdot (1-f)
\]

\[
R_{\text{end}} = \text{WHR} \cdot (1+f)
\]  

(16)
where, $CV_i$ is an alternate form of coefficient of variance, $\max(WHR_i)$ and $\min(WHR_i)$ are maximum and minimum width to height ratio respectively of signature samples of the $i$th person among total $N$ persons. By experimenting on a large set of signature samples, value of range_factor is obtained as 0.5.

In knowledge generation phase, WHR clusters of signature samples having close width to height ratio are formed while in Signature search and verification phase, matching cluster having WHR close to unknown signature sample is formed.
3.1.5 Blot-9, Blot-6, Blot-3 levels

To get the structural information of signature sample and its internal part, views are formed by dilating signature sample. As mentioned earlier, human being starts with global information going down to detailed information. As dilating is done, local details are lost and global structure becomes evident. It is as if the resolution is reduced or person is looking from a distance or with reduced attention. Two questions arise here: first, "How to do dilating?" and second, "To what extend dilating should be done?".

Dilating [35] is done by scanning a $3 \times 3$ window over the signature image. If five or more pixels out of the total nine pixels in $3 \times 3$ window are found on, all pixels in the window are made on. The second question is solved by visual examination of signature samples views as dilating is done repetitively. On investigation over a large sample signature samples, it is found that the signature samples at third, sixth and ninth level of dilating give reasonable structural information and are distinct from each other. Hence these three views are used in the proposed models. They are termed as blot-3, blot-6 and blot-9 where 'n' in 'blot-n' refers to the number of times dilation process is applied to get 'blot'. Figure 7 shows blots of genuine and forged signature samples of two different persons. Blot vectors are formed as shown in section 3.1.3. Figure 8 shows a signature sample with dilated image blot-0, blot-3, blot-6, blot-9 along with their 192-element vector representation.

For each signature sample in learning-set, all the three blots are formed. Mean of signature samples blot-3 vectors belonging to same persons are found. These mean vectors are stored in the knowledge base of blot-3. Similarly knowledge base of blot-6 and blot-9 are created.
Fig. 7: Blots of Genuine and Forged signature samples
Blot-0 sample signature

\[ \text{Rahul} \]

Blot-0 vector (16×12)

\[
\begin{align*}
0 & 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 2 \\
0 & 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 3 \\
0 & 2 2 0 0 0 0 0 2 0 0 0 0 0 0 3 \\
0 & 4 2 0 0 0 0 0 3 0 0 0 0 0 0 1 2 \\
2 & 3 2 0 0 6 0 0 3 0 0 0 0 0 0 2 1 \\
2 & 2 3 0 2 1 4 0 4 5 2 1 3 0 3 0 \\
3 & 8 1 0 2 0 5 0 6 3 4 0 4 0 4 3 \\
4 & 4 2 0 2 2 3 1 5 3 5 3 4 4 1 0 \\
5 & 0 3 0 2 4 5 4 1 0 0 1 0 0 0 0 \\
6 & 0 2 1 4 3 0 0 0 0 0 0 0 0 0 0 \\
5 & 0 0 3 2 0 0 0 0 0 0 0 0 0 0 0 \\
3 & 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
\end{align*}
\]

Blot-3 sample signature

\[ \text{Rahul} \]

Blot-3 vector (16×12)

\[
\begin{align*}
0 & 6 8 0 0 0 0 0 0 0 0 0 0 0 0 0 7 \\
0 & 8 8 1 0 0 0 0 4 2 0 0 0 0 0 1 7 \\
2 & 7 5 2 0 0 0 0 5 3 0 0 0 0 0 3 6 \\
3 & 6 6 1 0 5 1 0 7 2 0 0 0 0 4 5 \\
5 & 6 8 1 3 9 6 0 8 7 3 2 5 1 4 4 \\
6 & 8 8 0 5 8 9 1 9 9 6 3 7 1 7 3 \\
7 & 9 5 0 7 5 9 2 9 8 3 8 4 9 9 \\
8 & 9 6 0 7 7 8 6 8 7 9 8 9 9 6 5 \\
9 & 4 8 0 7 9 9 9 6 5 5 6 3 4 0 0 \\
9 & 0 7 7 8 8 3 2 0 0 0 0 0 0 0 0 \\
8 & 0 2 9 8 2 0 0 0 0 0 0 0 0 0 0 \\
6 & 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
\end{align*}
\]

Fig. 8 : Blots and their vector representation
Blot-6 sample signature

\[
\begin{array}{cccccccccccccccc}
0 & 9 & 9 & 3 & 0 & 0 & 0 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 3 & 9 \\
3 & 9 & 9 & 4 & 0 & 0 & 0 & 0 & 6 & 5 & 0 & 0 & 0 & 0 & 4 & 9 \\
5 & 9 & 9 & 4 & 0 & 1 & 0 & 0 & 8 & 5 & 0 & 0 & 0 & 0 & 6 & 9 \\
6 & 8 & 9 & 4 & 3 & 9 & 5 & 1 & 9 & 6 & 2 & 1 & 3 & 1 & 6 & 8 \\
8 & 9 & 9 & 3 & 6 & 9 & 9 & 5 & 9 & 9 & 5 & 8 & 4 & 8 & 7 & 9 \\
9 & 9 & 9 & 2 & 8 & 9 & 6 & 9 & 9 & 9 & 7 & 9 & 6 & 9 & 7 & 9 \\
9 & 9 & 8 & 1 & 9 & 9 & 8 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 2 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 6 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 1 \\
9 & 5 & 9 & 9 & 9 & 8 & 7 & 4 & 2 & 3 & 3 & 1 & 1 & 0 & 0 & 0 \\
9 & 2 & 8 & 9 & 9 & 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 0 & 1 & 4 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Blot-9 sample signature

\[
\begin{array}{cccccccccccccccc}
4 & 9 & 9 & 5 & 0 & 0 & 0 & 0 & 8 & 7 & 0 & 0 & 0 & 0 & 6 & 9 \\
6 & 9 & 9 & 6 & 0 & 0 & 0 & 1 & 9 & 7 & 0 & 0 & 0 & 0 & 7 & 9 \\
8 & 9 & 9 & 6 & 3 & 7 & 5 & 2 & 9 & 7 & 0 & 0 & 0 & 0 & 8 & 9 \\
9 & 9 & 9 & 6 & 7 & 9 & 8 & 7 & 9 & 8 & 7 & 8 & 6 & 9 & 9 & 9 \\
9 & 9 & 9 & 7 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 7 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 7 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 8 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 7 & 7 & 7 & 6 & 6 & 3 \\
9 & 8 & 9 & 9 & 9 & 9 & 9 & 6 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 4 & 7 & 9 & 9 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Fig. 8 : Blots and their vector representation (contd.)
3.1.6 Clustering

During knowledge generation phase of blot levels, distance threshold parameter for matching signature samples is decided. Signature samples having distance less than the threshold are put in the same cluster. Minmax algorithm [99] is used to find clusters, though experiments can be carried out with any other clustering algorithm.

The goal is to reduced the number of matching signature samples as one moves down the levels in a model. The threshold parameter is selected such that the clusters of signature samples should double at each level or in other words, number of elements in a cluster are reduced at each level, to get a possible binary tree. For example say at WHR level there are 10 clusters, then threshold of Blot-9 level should be such that about 20 clusters get created and threshold for Blot-6 level should be such that about 40 clusters get created and so on. On the current training set of signature sample, threshold values found are:

Blot 9 - 60, Blot 6 - 50, Blot 3 - 40

based on Euclidean distance 'D' between two n-dimensional vectors 'x' and 'y' as in (19):

\[ D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \]  

(19)

In signature search and verification phase these distance thresholds are used to find cluster which is matching the unknown signature sample.

The minmax clustering algorithm is given on the following page.
Minmax clustering algorithm

Input: Sample vectors

Output: Clusters of samples

Steps:

Step 1: Make first vector as cluster center ($Z_1$).

Step 2: Find vector farthest to $Z_1$. Make it cluster center ($Z_2$).

Step 3: Find distances between each cluster center and a remaining vector. Save the minimum of these distances.

Step 4: Repeat step 3 for all remaining vectors.

Step 5: Find the maximum distance between the minimum distances found in step 3 and step 4.

Step 6: If maximum distance is less than a threshold, Stop.

Step 7: Make the vector corresponding to maximum distance in step 5 as a new cluster center. Go to step 3.
3.1.7 Neural network level

Multiple neural networks are used for learning signature sample. Each of these neural network has 192 inputs nodes, 9 hidden nodes and 8 output nodes corresponding to eight persons. The number of Neural network that should be used and the persons whose samples should be given for training to a particular neural network is decided by the clusters obtained during the knowledge generation phase for WHR, Blot-9, Blot-6 and Blot-3. Persons whose signature samples are in the same cluster along a path of the cluster tree are kept for same neural network. It is seen that each person is associated with at least one neural network. Thus neural network will be able to distinguish those signature samples which are not distinguished by earlier levels.

For the given learning-set of signature samples, fourteen neural networks are determined from the cluster tree. This is obtained by moving up the cluster tree, starting from the clusters at Blot 3 level and forming groups of 8 persons which are found in same cluster along a path. Each group of 8 persons' signature samples will form training set for one Neural network. Each of these neural network is learned thrice independently. Training file contain vectors of signature samples without dilation (blot-0 vector) so that most detailed information is learned by neural network. For each person eight signature samples are used for training. Thus a training file for one neural network contains 64 vectors. The weights of so formed forty-two neural networks (fourteen network learned thrice) and the associated person’s identity numbers are stored in the NNW knowledge base. Person’s identity number is referred here as the reference number of the class of signatures belonging to a particular person. The neural network architecture used is given in figure 9 and the association of neural network and person’s identity number is listed on the following page for a set of 60 persons’ signature classes.
Fig. 9: Neural Network used by Signature Recognizer
## Association of Neural Network Number and Person’s Identity Number

<table>
<thead>
<tr>
<th>Neural Network No. 01</th>
<th>Person’s Identity Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>03 05 06 16 31 53 58 60</td>
</tr>
<tr>
<td>Neural Network No. 02</td>
<td>01 09 11 31 34 53 54 56</td>
</tr>
<tr>
<td>Neural Network No. 03</td>
<td>01 09 27 33 34 36 38 56</td>
</tr>
<tr>
<td>Neural Network No. 04</td>
<td>11 12 19 25 50 51 54 59</td>
</tr>
<tr>
<td>Neural Network No. 05</td>
<td>12 13 19 22 24 37 40 51</td>
</tr>
<tr>
<td>Neural Network No. 06</td>
<td>14 27 32 33 36 38 50 59</td>
</tr>
<tr>
<td>Neural Network No. 07</td>
<td>02 07 13 20 21 24 26 46</td>
</tr>
<tr>
<td>Neural Network No. 08</td>
<td>07 08 15 23 32 42 45 57</td>
</tr>
<tr>
<td>Neural Network No. 09</td>
<td>05 06 14 22 29 37 40 55</td>
</tr>
<tr>
<td>Neural Network No. 10</td>
<td>02 08 20 21 23 42 45 46</td>
</tr>
<tr>
<td>Neural Network No. 11</td>
<td>05 06 15 26 29 48 55 57</td>
</tr>
<tr>
<td>Neural Network No. 12</td>
<td>10 18 25 28 35 39 43 48</td>
</tr>
<tr>
<td>Neural Network No. 13</td>
<td>10 16 18 28 35 39 41 43</td>
</tr>
<tr>
<td>Neural Network No. 14</td>
<td>04 17 30 41 44 47 49 52</td>
</tr>
</tbody>
</table>
The Backpropagation algorithm [255], implemented by one of the authors, is used for training. The algorithm is given on the following pages. To speed up training process of neural network, the heuristic used is that the training should restart if:

- \((\text{total error} > 20 \text{ at epoch 200})\) or
- \((\text{total error} > 10 \text{ at epoch 600})\) or
- \((\text{total error} > 5 \text{ at epoch 1000})\) or
- 2000 epochs are completed.

This helps in completing training of a neural network at an average of 30 minutes on Pentium 75 MHz machine. Following parameters are used for each of the neural network:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Nodes</td>
<td>192</td>
</tr>
<tr>
<td>Hidden Nodes</td>
<td>9</td>
</tr>
<tr>
<td>Output Nodes</td>
<td>8</td>
</tr>
<tr>
<td>Learning factor</td>
<td>0.1</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>Tolerable Max. error</td>
<td>0.08</td>
</tr>
<tr>
<td>Weight adjustment</td>
<td>At end of Epoch</td>
</tr>
<tr>
<td>Initial Weight</td>
<td>Random</td>
</tr>
</tbody>
</table>

Tolerable maximum error of 0.08 is selected so that outputs '1' and '0' can vary from 0.6 through 1.0 and 0.0 through 0.4 respectively.
BackPropagation Algorithm

The Backpropagation algorithm consists of **Encoding** (supervised learning) stage and a **Recall** stage. A fully connected three layers (Input, Hidden, Output) feed-forward neural network is used as the topology.

**Terms:**

- \( i \) : input layer vector.
- \( h \) : hidden layer vector.
- \( o \) : output layer vector.
- \( d \) : desired output vector.
- \( e_o \) : error vector for output layer.
- \( e_h \) : error vector for hidden layer.
- \( W_{ih} \) : matrix of weights of links between input and hidden layers.
- \( W_{ho} \) : matrix of weights of links between hidden and output layers.
- \( \eta \) : learning factor
- \( \alpha \) : momentum factor
- \( F(x) \) : A sigmoid activation function

\[
\frac{1}{1+e^{-x}}
\]

**Encoding :-**

**Input** : Training vectors (Input vector, Desired Output vector)

**Output** : Weights of links
BackPropagation Algorithm (conid.)

Steps:

Forward Pass

Step 1: Randomize the weights of all links ($W_{ih}$ and $W_{ho}$).

Step 2: For a given input vector, compute hidden-layer neurons’ activation.

$$ h = F(i \cdot W_{ih}) $$

Step 3: Compute the output-layer neuron activation.

$$ o = F(h \cdot W_{ho}) $$

Backward Pass

Step 4: Compute the error vector between the desired vector and output layer vector.

$$ e_o = o(1-o)(o-d) $$

Step 5: Compute the error vector between the back propagated error vector and the hidden layer vector.

$$ e_h = h(1-h) \cdot W_{ho} \cdot e_o $$

Step 6: Modify the weights between hidden and output layers.

$$ W_{ho} = W_{ho} + \Delta W_{ho} $$

$$ \Delta W_{ho}(j) = (1-\alpha) \eta h e_o + \alpha \Delta W_{ho}(j-1) $$

where j and j-1 indicates current and previous iterations respectively.

Step 7: Modify the weights between input and hidden layers.

$$ W_{ih} = W_{ih} + \Delta W_{ih} $$

$$ \Delta W_{ih}(j) = (1-\alpha) \eta i e_h + \alpha \Delta W_{ih}(j-1) $$
BackPropagation Algorithm (contd.)

where \( j \) and \( j-1 \) indicates current and previous iterations respectively.

Step 8 : Repeat step 2 to step 7 on all training vectors.

Step 9 : Repeat step 2 to step 8, till the error between desired output vector and output vector is within tolerable limit for each training vector and for each neuron.

Recall :-

**Input** : Test vector

**Output** : Output vector

Step 1 : For a given test vector, compute hidden-layer neurons' activation.

\[
h = F(iW_{ih})
\]

Step 2 : Compute the output-layer neuron activation.

\[
o = F(hW_{ho})
\]
3.2 Numeral Recognition by Contour Tracing and Feature Extraction

3.2.1 The proposed technique

This research proposes a novel off-line technique based on very simple features, trying to merge both bottom-up and top-down approaches [72] followed by human beings to make fast decision. The proposed technique does not involve training as done by other methods. Numerals are dilated and converted to two-line contours. Contours are divided into segments which form four types of primitive features: horizontal segment, vertical segment, back slash and forward slash. The primitive feature have one of the two possible directions. Combination of these primitive features form four types of higher features: Right corner, Left corner, Top corner and Bottom corner. Contour also helps to find loops in a numeral. Test rules are formed using size information, loop information, primitive features and higher features. Tests also include fuzzy positional information for each feature. Each test has a weight associated with it depending upon its significance in recognizing a numeral. A test returns a success value quantifying the success obtained. Certainty factor for each numeral is calculated using weights associated with each test and success value returned by a test. Numeral with highest certainty factor is declared as recognized numeral. The flowchart on page 73, gives the major steps of the proposed techniques.
Flowchart for Numeral Recognition

Start

Input Image of a Numeral

Dilate Image

Form two-line contour

Segment contour

Form primitive features

Form higher features

Apply test rules

Output Numeral with highest Certainty Factor

Stop
3.2.2 Dilating, contouring and segmenting

Each numeral extracted from a series of numerals is dilated to remove noise caused due to the quality of paper, ink and the scanning process. Dilation is done by moving a $3 \times 3$ window over a numeral image. If five or more pixels out of the total nine pixels in window are found on, all pixels in the window are made on. Numerals in Figure 11(a) are shown dilated in Figure 11(b). Flowchart for dilating numeral is given on page 77.

Hilditch’s 8-connectivity number [252] is used with an alteration so that instead of getting a single-line thinned image, two-line contours of numerals are obtained. For each pixel, Hilditch’s 8-connectivity number $H_8$ is calculated using a $3 \times 3$ window centered at the pixel using the following equation (20):

$$H_8 = \sum_{i=1,3,5,7} \{if(w[i] = 0) then(w[i+1] \lor w[i+2]) endif\} \quad (20)$$

where $w[i]$ indicate whether pixel is 'on' or 'off' and $i$ is the cell position as mentioned in figure 10.

![Fig. 10: Cell positions for Hilditch’s connectivity number](image)

If Hilditch’s connectivity number is found zero for an 'on' pixel, the pixel is erased. This will delete all inner pixels, giving the outer contour of an image. Figures 11(c) show contours of numerals. For each numeral more than one contour can exist depending on a
Fig. 11: Dilation, contouring, segmentation of numerals
numeral and handwriting. For example numeral '8' will have two inner contours for the upper and bottom loops. Bad writing may have broken contours or joining of more than one contours. Flowchart for converting dilated numeral to double-line contour is given on page 78.

Reduction in the amount of information and elimination of noisy variation in numerals is done by dividing contour into line segments. Very small segment size gives more detail variation but along with it noise gets added. Big segment size eliminates noise but at the risk of losing significant information. Segment size is selected in such a manner that it depends on the size of a numeral.

In this research, segment size is in terms of number of points traversed in a contour. Number of points are taken as one-fourth of the numeral's height or width whichever is larger but at a junction or at an end of a contour, number of points are allowed to be one-twentieth. This is done to preserve information at the end of a contour. Figure 11(d) shows segmented contours. The flowchart for segmenting numeral, and forming it as a series of points is given on pages 79-84. If two or more segment are found to have same direction, that is if two or more segments have same sign of delta_x and also have same sign of delta_y, they are reduced to a single segment except for horizontal and vertical segments. Figure 11(e) shows numerals with reduced points.
Flowchart for 'dilating' numeral

Start

Reset flag for all pixels

i = 0

Get a 3X3 window surrounding the ith pixel

N

Are white pixel in window >= 5

Y

"flag" all pixels of window

i = i + 1

N

Are all pixels over?

Y

Make all flagged pixels white

Stop
Flowchart for 'contouring' numeral

1. Start
2. Set flag for all pixels of image
3. Get pixels of 3x3 windows centered at (x,y)
4. Is (x,y) a white pixel?
   - Yes: Get Hilditch's 8-connectivity number
   - No: Is connectivity number == 0?
     - Yes: Flag pixels(x,y) or to be erased
     - No: Detect pixels of image whose flags are reset
5. Stop
Flowchart for 'segmenting' numeral

Start

Y

Is height >= width?

segment_size = height / segment_ratio
small_segment_size = height / small_segment_ratio

N

segment_size = height / segment_ratio
small_segment_size = width / small_segment_ratio

Initialize indices for contour to 0

N

Are all rows scanned?

Go to next row

Y

Is white pixel present in current row?

Flag the current point as traced
segment point = 1
save point in contour array

Start tracing the current point

Get neighbouring 3x3 window points
Flowchart for 'segmenting' numeral (contd.)

- **A**
  - yespush=0
  - k=4
  - **N**
    - Is kth point of 3x3 window white?
    - **Y**
      - Push current point in stack
      - yespush++
      - k--
      - **Y**
        - Is k >= 1
        - **N**
          - k=8
          - B
Flowchart for 'segmenting' numeral (contd.)

- **D**
  - **N**: Is yespush > 0?
  - **Y**: Get a point from top of the stack
    - **segment_points++**
    - **Is segment_points == segment_size**
      - **Y**: segment_point = 1
      - **T**: Save point in contour array
      - **R**: End
      - **N**: Is it a junction?
        - **N**: Is segment_points >= segment_size
          - **N**: End
          - **Y**: Go back to get a point from top of the stack
Flowchart for 'segmenting' numeral (contd.)

T

Is more than one point in stack? 

N

R

Y

Are more than one points stored in current iteration of contour tracing?

N

remove saved points

Y

store contour array index in breakpoint array
Store breakpoint type as BP_JUNCTION

save point in contour array as a start of new series

R

F

Extract features of character

Return recognized characters

stop
Flowchart for 'segmenting' numeral (contd.)

Is \( \text{segment} \_ \text{points} \geq \text{small} \_ \text{segment} \_ \text{size} \)?

- Y: save point in contour array

- N:
  - Are more than one point stored in current iteration of contour tracing?
    - Y: store contour array index in breakpoint array, store breakpoint type as BP\_END\_STACK
    - N: get a point from top of stack
      - segment points = 1
      - save points in contour array as a start of new series

Flag the current point as traced
3.2.3 Fuzzy positional information and size

Each line segment, obtained after contouring, is allotted one of the sixteen possible fuzzy positions. Depending on the x-coordinate of endpoints of a segment, it can occupy one of the four possible horizontal positions: Left, Right, Middle and Full over the width. Similarly depending on the y-coordinate of endpoints of a segment, it can occupy one of the four possible vertical positions: Top, Bottom, Middle and Full over the height. Figure 12 show these positions considering a line segment with coordinates \((x_1,y_1)\) and \((x_2,y_2)\) where \(x_1 < x_2\) and \(y_1 < y_2\). The range of values taken by \(x_1\), \(x_2\) with respect to width and \(y_1\), \(y_2\) with respect to height is indicated by arrows.

![Diagram](image)

**Fig. 12 : Fuzzy position of a line segment**

A four bit code (LRTB) is used to store sixteen position. The first two bits store horizontal position and next two bits store vertical information. The bit value is calculated as:
L = ((x1 <= width/2) && (x2 <= 3*width/4))
R = ((x1 > width/4) && (x2 > width/2))
T = ((y1 <= height/2) && (y2 <= 3*height/4))
B = ((y1 > height/4) && (y2 > height/2))

If both L and R bits are 1, it indicate middle horizontal position and if both L and R bits are 0, it indicates full horizontal position. Similarly middle and full vertical position are determined based on both 1 or both 0 values of T and B bits respectively. The sixteen positions are abbreviated as LT, LM, LB, LF, MT, MM, MB, MF, RT, RM, RB, RF, FT, FM, FB and FF.

The size information relative to height and width is found for a line segment and is normalized in the scale of 0 to N. Following equation (21) shows calculation of size in x and y direction respectively.

\[
\begin{align*}
size_x &= ((x_{i+1} - x_i) \times N) / Width \\
size_y &= ((y_{i+1} - y_i) \times N) / Height
\end{align*}
\]

3.2.4 Feature extraction

The process of feature extraction by the proposed technique includes extraction of loops, primitive features and higher features. The basic steps for feature extraction are given in flowchart on page 87.

**Primitive features**

The process of extracting, dilating, contouring and segmenting gives line segments. These line segments are classified into primitive features based on slope and direction as shown in Table 1.

The occupancy of these features for SLOPE_CONSTANT = 5 is given in figure 13. Figure 14 shows primitive features.
Flowchart for 'feature-extraction' of numerals

Start

Are no of points \( \leq 1 \)

\( \text{Y} \)

Return ' ' as no character present

\( \text{N} \)

Stop

Remove intermediate points

Find loops

Merge series

Detect primitives features

Merge feature

Form higher feature

Next series

Are all series over

\( \text{Y} \)

Fill information in SYMBATTRIB

\( \text{N} \)

Find character

Stop
If the consecutive primitive features are found to be same, they are merged as one by combining their attributes: position, size\(_x\) and size\(_y\). For primitive features BD, BU, FD and FU, all merging features should have either gentle slope (\(\delta_x \geq \delta_y\)) or all should have sharp slope (\(\delta_x \leq \delta_y\)). This is done to avoid merging of features in different direction. The flowchart for extracting primitive features is given on page 90.

Fig. 13: Slopes of primitive features

Fig. 14: Primitive features
### Table 1: Primitive features

<table>
<thead>
<tr>
<th>Primitive Feature</th>
<th>Type</th>
<th>Direction</th>
<th>Slope (Modulus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>Horizontal</td>
<td>Right</td>
<td>&lt; (1/SLOPE_CONSTANT)</td>
</tr>
<tr>
<td>HL</td>
<td>Horizontal</td>
<td>Left</td>
<td>&lt; (1/SLOPE_CONSTANT)</td>
</tr>
<tr>
<td>VD</td>
<td>Vertical</td>
<td>Down</td>
<td>&gt; SLOPE_CONSTANT</td>
</tr>
<tr>
<td>VU</td>
<td>Vertical</td>
<td>Up</td>
<td>&gt; SLOPE_CONSTANT</td>
</tr>
<tr>
<td>BD</td>
<td>BackSlash</td>
<td>Down</td>
<td>&gt; = (1/SLOPE_CONSTANT) and &lt;= SLOPE_CONSTANT</td>
</tr>
<tr>
<td>BU</td>
<td>BackSlash</td>
<td>Up</td>
<td>&gt; = (1/SLOPE_CONSTANT) and &lt;= SLOPE_CONSTANT</td>
</tr>
<tr>
<td>FD</td>
<td>ForwardSlash</td>
<td>Down</td>
<td>&gt; = (1/SLOPE_CONSTANT) and &lt;= SLOPE_CONSTANT</td>
</tr>
<tr>
<td>FU</td>
<td>ForwardSlash</td>
<td>Up</td>
<td>&gt; = (1/SLOPE_CONSTANT) and &lt;= SLOPE_CONSTANT</td>
</tr>
</tbody>
</table>

### Table 2: Higher features

<table>
<thead>
<tr>
<th>Higher Feature</th>
<th>Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>Right Corner</td>
<td>Down</td>
</tr>
<tr>
<td>RU</td>
<td>Right Corner</td>
<td>Up</td>
</tr>
<tr>
<td>LD</td>
<td>Left Corner</td>
<td>Down</td>
</tr>
<tr>
<td>LU</td>
<td>Left Corner</td>
<td>Up</td>
</tr>
<tr>
<td>BR</td>
<td>Bottom Corner</td>
<td>Right</td>
</tr>
<tr>
<td>BL</td>
<td>Bottom Corner</td>
<td>Left</td>
</tr>
<tr>
<td>TR</td>
<td>Top Corner</td>
<td>Right</td>
</tr>
<tr>
<td>TL</td>
<td>Top Corner</td>
<td>Left</td>
</tr>
</tbody>
</table>
Flowchart to extract primitive features of numerals

1. Start
2. Is series forming loop? (Y/N)
   - Y: formloop=1
   - N: formloop=0
3. Find series attributes
4. Find deltX and deltY for each successive pair of points
5. Find horiz. or vert. type primitives feature
6. Is type valid? (Y/N)
   - Y: Find forward of backward slash type primitive feature
   - N: Go to next deltX, deltY
7. Are all deltX deltY over? (Y/N)
   - Y: Stop
   - N: Go to next deltX, deltY
Higher features

Higher Features are formed by combining two or three primitive features found in a sequence. As Primitive features, Higher features are also classified based on type and direction as shown in Table 2.

Valid combinations of primitive features are stored in a look-up table to form higher features. Figure 15 shows formation of 'RD' Higher feature by nine combinations of three primitive features and three combinations of two primitive features. Similarly other higher features are formed from primitive features as shown in figure 16. Each higher feature has size_x, size_y, and positional attributes obtained by combining attributes of constituent primitive features. The process of forming higher feature is shown in flowchart on page 93-94. Table 3 is the look-up table for higher feature.

Fig. 15 : Higher feature - Right corners
Fig. 16: Higher features - Left, Top and Bottom corners
Flowchart to extract higher features of numerals

Start

- Initialize index for higher features, k=0

- Initialize index for primitive features, i=0

  - Is i+2 < no of prime features? N → Find higher features for last two primitive features. Consider first primitive feature if loop.

  - Y → digits2=feature[i].type*10+feature[i+1].type
digit3= digits2*10+feature[i+2].type

    - Reset flag 3-digit feature
    - feature=GetDigits3feature()

    - Is feature valid? N → feature=GetDigits2feature()

    - Y → Combine rectangle of three features

      - Set flag 3-digit feature

    - Is feature valid? N → Combine rectangle of two features

Stop
Flowchart to extract higher features of numerals (contd.)

A

Is feature valid?

Y

Store kth higher feature attribute

Higher feature[k],
prim_feature_index1 = i
prim_feature_index2 = i + 1

N

Is flag3digits feature set?

Y

Higher feature[k].primitive_index3
= i + 2

N

Higher feature[k].primitive_index3
= -1

k = k + 1

B
Table 3: Higher Feature Look-up table

#define NOOFDIGITS2 24 //Total two-digits code for finding features
#define NOOFDIGITS3 72 //Total three-digits code for finding features

typedef struct tagFSTORE {
    int code; // code of sequential primitive features
    int h_feature; // higher feature formed by primitive features
} FSTORE; // High Level Feature look-up table entry

// enumeration of primitive feature and higher features
enum {
    ND=0,
    HR, HL, VD, VU, BD, BU, FD, FU, // Primitive features
    RD=11, RU, LD, LU, BR, BL, TR, TL // Higher features
};

// Store 3-digits codes for features in ascending order
FSTORE digits3store[NOOFDIGITS3] = {
    {132, RD}, {136, RD}, {137, RD}, {142, RU}, {146, RU}, {147, RU},
    {231, LD}, {235, LD}, {238, LD}, {241, LU}, {245, LU}, {248, LU},
    {314, BR}, {316, BR}, {318, BR}, {324, BL}, {326, BL}, {328, BL},
    {413, TR}, {415, TR}, {417, TR}, {423, TL}, {425, TL}, {427, TL},
    {514, BR}, {516, BR}, {518, BR}, {524, BL}, {526, BL}, {528, BL},
    {532, RD}, {536, RD}, {537, RD}, {542, RU}, {546, RU}, {547, RU},
    {613, TR}, {615, TR}, {617, TR}, {623, TL}, {625, TL}, {627, TL},
    {631, LD}, {635, LD}, {638, LD}, {641, LU}, {645, LU}, {648, LU},
    {714, BR}, {716, BR}, {718, BR}, {724, BL}, {726, BL}, {728, BL},
    {731, LD}, {735, LD}, {738, LD}, {741, LU}, {745, LU}, {748, LU},
    {813, TR}, {815, TR}, {817, TR}, {823, TL}, {825, TL}, {827, TL},
    {832, RD}, {836, RD}, {837, RD}, {842, RU}, {846, RU}, {847, RU}
};

// Store 2-digits codes for features in ascending order
FSTORE digits2store[NOOFDIGITS2] = {
    {16, RU}, {17, RD}, {25, LD}, {28, LU},
    {36, BL}, {38, BR}, {45, TR}, {47, TL},
    {52, RD}, {54, BR}, {57, RD}, {58, BR},
    {61, LU}, {63, TL}, {67, TL}, {68, LU},
    {71, LD}, {74, BL}, {75, LD}, {76, BL},
    {82, RU}, {83, TR}, {85, TR}, {86, RU}
};
**Loop information**

The loop information is obtained in two ways. The first method looks at the start and end points of contours of a numeral. If these points are close to each other, it indicates presence of a loop. Rough handwriting may have many broken contours. These necessitates joining of contours before finding a loop.

The second method is for incomplete loops. Figure 17 shows numerals with such loops. The loop is detected by looking at higher features and if three of the four possible higher features are found in a sequence, a loop is considered to be existing. The simplest case is the sequential existence of Left corner, Bottom corner, Right corner and Top corner in a contour. If any three of these corners exist in sequence, there is a possibility of loop formation. For every loop found, the relative size and positional information is stored.

![Incomplete inner loop](image)

Fig. 17 : Incomplete loops in numerals
3.2.5 Test rules

Test rules are formed using numeral size information, loop information, primitive
features and higher features. Fuzzy positional information, size $x$ and size $y$ for each feature
are also used to form tests. Four typical tests are shown below:

(i) Existence of Top and Bottom Loop
(ii) Existence of HX at FT
(iii) Existence of RX at XB or XF
(iv) Existence of RX,LX,RX in sequence at !LX

Test (i) is for numeral 8, test 2,3 are for numeral 5 and test 4 is for numeral 3. Type HX
indicate primitive feature HR or HL. Type RX indicate higher feature RD or RU. Type LX
indicate higher feature LD or LU. Position XB indicate either of LB,RB,MB or FB bottom
positions. Position LX indicate either of LT, LB,LM or LF Left positions. Exclamation
before Position in !LX indicate position not on Left side. Tests used for numeral and their
descriptions are listed on pages 99-102.

Each test has a weight associated with it depending upon its significance in
recognizing a numeral. The weight of a test is maximum if it is a test for a single numeral.
The weight of a test reduces if it is required for more numerals because the success of the
test indicate possibility of more than one numeral. Thus every test is for one or more
numerals and has associated weight. The weight of ith test $w_i$ is found as in (22):

$$w_i = \frac{S_i}{S_i}$$

(22)

where $S_i$ is total symbols and $S_i$ is the number of symbols checked by the ith test. The weight
associated with each test is shown in table 4.

A test returns a success value between 0 and $S$ quantifying the success obtained.
Value of 'S' indicate full success, '0' indicate no success while in between values indicate partial successes. The success value is determined using size_x and size_y attributes of loops, primitive features and higher features.

Certainty factor for each numeral, to be a possible choice for a numeral under test, is calculated by the following equation (23):

\[
C.F._j = \frac{\sum_{i} T_{i,j} \ast w_i \ast S_i}{\sum_{i} T_{i,j} \ast w_i}
\]

(23)

where, C.F._j is the certainty factor for jth numeral, \( \Sigma_i \) is the summation over all tests, \( T_{i\cdot j} \) is 1 if ith test exists for jth numeral else it is 0, and \( S_i \) is the success value returned by ith test.

Once Certainty factor for all possible numerals are calculated, the numeral with highest certainty factor is declared as a recognized numeral. But, if the highest certainty factor is below a threshold, the numeral under test is treated as cannot-be-recognized. The flow chart for testing numerals is given on page 104.
### Tests for Numeral Recognition

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tested (Feature @ Position)</th>
<th>Success value depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size test</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 Any @ Any</td>
<td>height,width</td>
</tr>
<tr>
<td><strong>Loop tests</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8 Loop @ (!Full &amp;&amp; Top &amp;&amp; Bottom)</td>
<td>__________</td>
</tr>
<tr>
<td>3</td>
<td>9 Loop @ (!Full &amp;&amp; Top &amp;&amp; !Bottom)</td>
<td>________</td>
</tr>
<tr>
<td>4</td>
<td>6 Loop @ (!Full &amp;&amp; !Top &amp;&amp; Bottom)</td>
<td>________</td>
</tr>
<tr>
<td>5</td>
<td>4 Loop @ (!Full &amp;&amp; (Top</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8' Loop @ (!Full &amp;&amp; (Top</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 Loop @ Full</td>
<td>________</td>
</tr>
<tr>
<td>8</td>
<td>1,2,3,5,7,4',8' ILoop @ Any</td>
<td>________</td>
</tr>
<tr>
<td><strong>Primitive feature tests</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5,7 N_HX @ FT</td>
<td>size_x</td>
</tr>
<tr>
<td>10</td>
<td>1 N_VX @ XF</td>
<td>size_y</td>
</tr>
<tr>
<td>11</td>
<td>7 N_VX @ !LX</td>
<td>size_y</td>
</tr>
<tr>
<td>12</td>
<td>4' N_VX @ LT</td>
<td>size_y</td>
</tr>
<tr>
<td>13</td>
<td>4' N_VX @ RB</td>
<td>size_y</td>
</tr>
<tr>
<td>14</td>
<td>2 (N_HX @ XB)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7 !(N_HX @ XB)</td>
<td>size_x</td>
</tr>
</tbody>
</table>
## Tests for Numeral Recognition (contd.)

<table>
<thead>
<tr>
<th>Test Sr. No.</th>
<th>Numerals Tested</th>
<th>Tests (Feature @ Position)</th>
<th>Success value depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>Any @ Any</td>
<td>rect_of_4</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>Any @ Any</td>
<td>rect_of_4</td>
</tr>
</tbody>
</table>

* Higher features tests

<table>
<thead>
<tr>
<th>Test Sr. No.</th>
<th>Numerals Tested</th>
<th>Tests (Feature @ Position)</th>
<th>Success value depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>0,6</td>
<td>LX @ (LF</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>!LX @ (LF</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>LX @ (LT</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>!(LX @ (LT</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>2,3,5,6,8,8'</td>
<td>BX @ XB</td>
<td>size_x</td>
</tr>
<tr>
<td>23</td>
<td>4',7</td>
<td>!(BX @ XB)</td>
<td>size_x</td>
</tr>
<tr>
<td>24</td>
<td>8'</td>
<td>BX @ XT</td>
<td>size_x</td>
</tr>
<tr>
<td>25</td>
<td>5,6</td>
<td>!(BX @ XT)</td>
<td>size_x</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>RX @ (XB</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>RX @ (XT</td>
<td></td>
</tr>
</tbody>
</table>

* Sequential higher features tests

<table>
<thead>
<tr>
<th>Test Sr. No.</th>
<th>Numerals Tested</th>
<th>Tests (Feature @ Position)</th>
<th>Success value depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>3</td>
<td>(((RU1-LU-RU2) &amp;&amp; (RU1 == RU2))</td>
<td></td>
</tr>
<tr>
<td>Sr. No.</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Compute Height to Width ratio.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Check for a loop at top and bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Check for a loop at top position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Check for a loop at bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Check for a loop at top or mid position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Check for a loop at top or bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Check for a loop at full position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Check for absence of loops.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Check for a nearly horizontal line at full-top position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Check for a nearly vertical line at left-full, middle-full or right-full position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Check for a nearly vertical line which is not at left position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Check for a nearly vertical line at left-top position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Check for a nearly vertical line at right-bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Check for a nearly horizontal line at bottom position or a back slash at bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Check for absence of nearly horizontal line at bottom position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Check for presence of a projecting rectangle at extreme right-middle position of numeral 4.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Tests Description

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Check for absence of a projecting rectangle at extreme right-middle position of numeral 4.</td>
</tr>
<tr>
<td>18</td>
<td>Check for a left corner at left-full or full-full position.</td>
</tr>
<tr>
<td>19</td>
<td>Check for absence of left corner at left-full or full-full position.</td>
</tr>
<tr>
<td>20</td>
<td>Check for left corner at left-top or full-top or left-bottom or full-bottom position.</td>
</tr>
<tr>
<td>21</td>
<td>Check for absence of left corner at left-top or full-top or left-bottom or full-bottom position.</td>
</tr>
<tr>
<td>22</td>
<td>Check for bottom corner at bottom position.</td>
</tr>
<tr>
<td>23</td>
<td>Check for absence of bottom corner at bottom position.</td>
</tr>
<tr>
<td>24</td>
<td>Check for bottom corner at top position.</td>
</tr>
<tr>
<td>25</td>
<td>Check for absence of bottom corner at top position.</td>
</tr>
<tr>
<td>26</td>
<td>Check for right corner at bottom position or full position.</td>
</tr>
<tr>
<td>27</td>
<td>Check for right corner at top or full top position.</td>
</tr>
<tr>
<td>28</td>
<td>Check a sequence of: right corner - left corner - right corner, either all three corners going up or all three corners going down. The vertical size of both the right corners should be nearly equal.</td>
</tr>
</tbody>
</table>
### Table 4: Weight table for tests and associated symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Test</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Test 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Test 3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Test 4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Test 5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>Test 6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Test 7</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>Test 8</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Test 9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 10</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>Test 11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Test 12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>Test 13</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 14</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 17</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 18</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 19</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>Test 21</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 22</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Test 23</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>Test 24</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>Test 25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 26</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Test 27</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>Test 28</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sum Weight: 18 26 26 40 24 28 26 38 14 24 32 38
Flowchart to 'test' numerals

Start

i=0

Are all tests executed?

Y

Execute ith test and store success value

N

i=i+1

Find certainty value for each symbol

Find symbol with maximum certainty

Print success value of test and certainty value of symbols

Y

Is max certainty >= threshold

N

Return symbol associated with max certainty

Return symbol 'I'

Stop
3.3 Character Recognition by Dilation and Centroid-based Normalization

3.3.1 Introduction

Most of the documents which include handwritten characters have either isolated characters or connected character (cursive script). For example, applications such as income-tax form or share form deal with independent characters and these characters are asked to be written in capital. While in other applications such as bank 'cheque' reader allow cursive script involving both small and capital characters. In a cursive script, character segmentation is a problem. Even for a human reader to extract character in isolation of the knowledge about the word is a difficult task.

The research looks for application involving isolated capital characters and numerals by proposing a novel technique to achieve results comparable to other researches. The method can also supplement cursive word recognition by finding confidence values for matching characters.

3.3.2 The proposed technique

Template matching of handwritten characters is difficult because of varied style of writing a same character. But template matching can be done, if one can elastically pull various significant point of character and match with the template. How to do this elastic pulling? How much elastic pulling should be done? How much computation load will be there? - are some of the questions which need to be pondered upon.

Characters of the same class have many small variation but their overall shape remains same which can help them to distinguish from other character classes. Each character class can be represented by a jacket. Jackets should be such that it can fit in characters of the same class but small enough to reject samples of other classes. Character recognition can then be considered as finding the best-fit jacket for an unknown character.
sample. How well a character sample fit in a jacket or in other words how much elastic pull is needed to fit a sample in a jacket, determines an unknown character. The proposed technique is based on this idea.

Dilation of character is done to form jacket of character sample. Centroid of dilated character is found. This centroid is used to divide the rectangular extent of the character into four regions. Figure 18 shows these steps. Each of these four regions are normalized to 16-element vector, thus giving a 64-element vector for a character. Dilated training samples' vectors are used to compute mean and variance of each character class. During testing, distance calculations are done based on the mean and variance of each class. Confidence values for each detected character class is derived based on its distance from the character sample under test.

<table>
<thead>
<tr>
<th>Original Character</th>
<th>Dilated Character</th>
<th>Vector Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⇒ A</td>
<td>0008 7000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0039 9700</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0087 5910</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0495 4940</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0895 4860</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2950 0591</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7910 0078</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9600 0028</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 18: Processing a character sample
3.3.3 Character dilation and centroid-based normalization

Character dilation

To get the structural information of a character sample, the sample is dilated. As dilating is done, minor variation and noise are lost and global structure becomes evident. Character samples can have various size. Dilating a small size character can make it lose its structural information while dilating, once, a large size character may not even remove noise. Hence the dilation is made adaptive to character size; larger the character, more the number of dilation cycles. Dilating [35] is done by scanning a $3 \times 3$ window over the character image. If five or more pixels out of the total nine pixels in $3 \times 3$ window are found on, all pixels in the window are made on. Fig. 19 shows some dilated characters.

![Fig. 19: Dilated character samples](image-url)
Centroid based normalization

The black and white image of character is normalized to a grid of $8 \times 8$ cells. The process of converting an arbitrary size character height and width to $8 \times 8$ is shown in the flowchart on pages 109-110. For each cell, the gray level is calculated as in (24):

$$\text{GrayLevel}_k = \frac{\sum_{i=1}^{m_k} \sum_{j=1}^{n_k} I_{i,j}}{m_k \times n_k} \times (L-1) \tag{24}$$

where, Gray Level$_k$ is the gray level of the kth cell, $m_k$ and $n_k$ are number of pixel rows and columns in the kth cell respectively, $I_{i,j}$ is the black/white intensity of a pixel in a cell and $L$ is the number of gray shades. Before normalization, the centroid $(x_c, y_c)$ of the dilated character is calculated as in equation (25) with pixel intensity $p(x_i, y_i)$ at point $(x_i, y_i)$.

$$x_c = \frac{\sum_{j=1}^{\text{height}} \sum_{i=1}^{\text{width}} x_i \times p(x_i, y_i)}{\sum_{j=1}^{\text{height}} \sum_{i=1}^{\text{width}} p(x_i, y_i)}$$

$$y_c = \frac{\sum_{j=1}^{\text{height}} \sum_{i=1}^{\text{width}} y_i \times p(x_i, y_i)}{\sum_{j=1}^{\text{height}} \sum_{i=1}^{\text{width}} p(x_i, y_i)} \tag{25}$$

The centroid divides the rectangular extent of character into four regions. Each of these regions is normalized into a $4 \times 4$, 16-element vector. Thus a normalized character sample is stored as vector with 64 elements, with each element having a value between 0 to $L-1$.

Centroid based normalization helps to compensate extra extent of character. Fig. 19 shows centroid of dilated characters by 'white' cross lines. In fig. 19, the first 'R' has an extended right stroke compared to second 'R' but the centroid of first and second 'R' is nearly at the same structural position. Hence using centroid, instead of a direct center point, helps for better template matching.
Flowchart for normalizing character size

Start

Input normal_value and total_value

value = total_value / normal_value

remainder = total_value % normal_value

i = 0

Let p_cut stores cuts in total value to get normal value

p_cut[i] = value * i

i++

Y

Is i < normal_value?

N

Is remainder_value = 0?

Y

fract_increment = normal_value / remainder_value

N

i = 1

A

B
Flowchart for normalizing character size (contd.)

1. \[ \text{index} = \text{fract increment} \times i + 0.5 \]
2. \[ \text{index} = 0? \]
   - Yes: \[ \text{index}++ \]
   - No: \[ j = \text{index} \]
3. \[ p_{\text{cut}}[j]++ \]
   - Yes: \[ j < \text{normal value} + 1? \]
     - Yes: \[ \text{index}++ \]
     - No: \[ i < \text{remainder value} + 1? \]
       - Yes: \[ \text{index}++ \]
       - No: \[ \text{Stop} \]
4. \[ B \]
5. \[ \text{Stop} \]
3.3.4 Distance and confidence value

Vectors of the dilated training samples of characters are used to find mean and variance of different character classes. Mean 'm' and variance 'v' are computed as in equation (26) and (27) respectively, where 'N' indicate the number of samples present in a character class and 'value' indicate the vector element value for the new training sample.

\[ m_{\text{new}} = \frac{m_{\text{old}} \cdot N + \text{value}}{N + 1} \]  \hspace{1cm} (26)

\[ v_{\text{new}} = \left( \frac{(v_{\text{old}} + m_{\text{old}}^2) \cdot N + \text{value}^2}{N + 1} \right) - m_{\text{new}}^2 \]  \hspace{1cm} (27)

where \( m_{\text{old}} \) and \( v_{\text{old}} \) are present mean and variance value while \( m_{\text{new}} \) and \( v_{\text{new}} \) are mean and variance after the new training sample is added to the corresponding class. The learned values of mean and variance for different classes are used for testing an unknown character sample.

The unknown character sample is dilated, its centroid computed and then is normalized to a vector. The vector of the dilated unknown character is compared with each of the character classes' vectors using the distance formula given in equation (28).

\[ d = \sqrt{\sum_{i=1}^{n} \frac{(\text{value}_i - m_i)^2}{v_i}} \]  \hspace{1cm} (28)

where 'd' is the distance; 'n' is the length of vectors; \( m_i \) and \( v_i \) are the ith vector element of mean and variance vectors respectively of a trained character class and \( \text{value}_i \) is the ith element value of the unknown character vector.

Confidence value determination of characters can help cursive word recognition algorithms. One method of calculating confidence value is to determine : (i) how close a sample is to the nearest character class and (ii) how close a sample is to other nearer classes.

To achieve this, three character classes closest to an unknown character sample are
determined. The distances of first, second and third character classes are denoted as 'f', 's' and 't'. The difference in the distances with second and first character class is denoted as 's_f' while the difference in the distances with third and second character class is denoted as 't_s'. The confidence values $CV_f$, $CV_s$ and $CV_t$ showing that the test character sample belong to the first, second and third character class respectively are given as in equation (29):

$$
CV_f = 100 - \alpha \ast (\beta_1 \ast f - \beta_2 \ast s_\_f - \beta_3 \ast t_\_s)
$$

$$
CV_s = 100 - \alpha \ast (\beta_1 \ast s + \beta_2 \ast s_\_f - \beta_3 \ast t_\_s)
$$

$$
CV_t = 100 - \alpha \ast (\beta_1 \ast t + \beta_2 \ast s_\_f + \beta_3 \ast t_\_s)
$$

where, $\alpha$, $\beta_1$, $\beta_2$, $\beta_3$ are the constants to range the confidence values. The confidence values are clipped to be within the range of '0' through '100'.

Flowcharts for training and testing of character sample by this approach are given on page 113 and page 114 respectively.
Flowchart for 'training' using character samples

1. Start
2. Input character 'Image' and its 'class'
3. Dilate character based on size
4. Compute centroid of diluted image
5. Normalize to vector using centroid
6. Update mean and variance of given class
7.Knowledge Base
8. Stop
Flowchart for 'testing' using character samples

Start

Input character 'Image'

Dilate character based on size

Compute centroid of dilated image

Normalize to vector using centroid

Compute distance from all classes

Output character with maximum confidence

Stop

Knowledge Base
3.4 Cursive words and phrase recognition using fuzzy features and contextual information

3.4.1 Introduction

Human, while reading, normally read word as a whole rather than character by character reading. He forms a dynamic vocabulary of limited words based on the subject he is reading. This dictionary helps to read words as a whole even if they are partial or misspelled. Even human, perhaps, guess [257] the next word(s) and then just checks his hypothesis by comparing the perceived word image with the expected word(s) or by actively looking for certain words in the text. Only when human cannot understand a word, he goes into its details. Feedback is performed between different levels and hence decision may be reversible [258].

One can say that the top-down process is controlled by regular information and the bottom-up process by singular information. One of the main difficulties with such an approach is the extraction of high-level contextual information from a text. However, there is a certain number of well-defined task like reading postal address or banking cheques where input information is highly structured and general context is known. For these tasks word-based recognition using regular/singular features seems quite appropriate [259]. It is now generally accepted that high performance of handwritten words will only be achieved by the use of the context.

3.4.2 The proposed technique

An off-line method for handwritten text recognition is proposed. It is a hybrid approach using both holistic and analytical strategies. The following questions were pondered upon while developing this technique.
* What features to take?
* How to extract features?
* How to find reference horizontal lines?
* How to segment a word into vertical segments?
* How to represent incomplete features?
* How to use positional information of features?
* How to combine values of various features to get a matching word?
* How to structure a dictionary?
* How to separate words in a phrase?
* How to use contextual information to get a matching phrase?
* What are the constrains for text writing?

Flowchart on page 117 gives the steps of the proposed technique.

3.4.3 Fuzzy features

To have holistic view of a word, certain prominent features in a word need to be selected. Normally a human concentrates on first character of a word and looks for feature such as upper strokes, lower strokes and loops. These features are quite evident even in a bad handwriting. Hence upper stroke, lower stroke, middle loop are taken to represent global view of a word while the first character is analytically determined.

The extraction of features required a word to be sectioned into three parts: upper, middle and lower regions by finding out two horizontal reference lines limiting the middle region. This is done using horizontal line pixel density graph as shown in fig. 20. Using thresholds, as fractions of the peak density gives the location of two reference lines.

The word is then partitioned into vertical segments using vertical line pixel density graph as shown in fig. 20. Peaks and valleys across the graph are used to set thresholds for
Flowchart for phrase recognition

Start

Input unknown phrase image

Read Text Dictionary

Read Word Dictionary

Find total words in input phrase

Find features for word

Find confidence value of matching words using 'Word' Dictionary

Are all words in input phrase over?

Y

Find confidence value of matching texts using 'Text' Dictionary

Output text with maximum C.V.

Stop
segmentation. The middle region is further cleaved into top, bottom, left, right and middle regions.

Each segment is scanned to detect the presence of features. Upper stroke and lower stroke are looked for in the upper and lower region respectively, provided the upper and lower region has reasonable height compare to middle zone. Middle loop is looked for in the middle region of a segment.

For extracting the first character more than one initial segment is scanned as the character may span more than one segment. The segments’ width information and vertical density of the segments help to extract the first character. The flowchart for extracting features from a word is shown on page 119.

Each feature has different strength depending upon the writing style and variation in written word. Even the algorithms which extract these features have their limitations in correctly detecting the features. To take into account this inexactness, fuzzy values are assigned to features.

![Word segmentation](image)

Fig. 20 : Word segmentation
Flowchart for extracting features from word sample

1. Start
2. Input word image
3. Find horizontal reference lines
4. Cleave word with vertical segment
5. Extract first character
6. Find confidence value of first character
7. Find upper stroke, lower stroke, middle loop in a vertical segment
8. Find confidence value of upper stroke, lower stroke and middle loop
9. Are all vertical segment over?
   - N: Go back to step 5
   - Y: Stop
The fuzzy value of upper (FV_{us}) and lower stroke (FV_{ls}) is found as in equation (30) while that of middle loop (FV_{ml}) is found by equation (31).

\[
FV_{us} = \frac{L_{vs}}{H_u}, \quad FV_{ls} = \frac{L_{vs}}{H_l}
\]

\[
FV_{ml} = 0.2 \times \left( \frac{L_{vs}}{H_{mt}} + \frac{L_{hs}}{H_{mb}} + \frac{L_{hs}}{W_{ml}} + \frac{B_{nm}}{T_{nm}} \right) \times f
\]

where \( L_{vs}, L_{hs} \) are the length of vertical and horizontal stroke; \( H_u, H_l, H_{mt}, H_{mb} \) are the height of upper, lower, middle top and middle bottom regions; \( W_{ml}, W_{mr} \) are the width of middle left and middle right regions; \( B_{nm} \) and \( T_{nm} \) are count of off pixels and total pixels in middle-middle region respectively. Factor \( 'f' \) is 1.0 if middle region is a square, otherwise the value of \( 'f' \) is lesser than 1.0 depending upon the height and width of the rectangle.

Each word is characterized by the instances and position of upper strokes, lower strokes and middle loops. The instance of a feature is said to be existing if its fuzzy value is above a threshold. \( P_{-FV} \), the fuzzy value based on relative position of an instance in an unknown word and a reference word is found as in equation (32).

\[
P_{-FV} = 1.0 - \left| \frac{P_{un}}{Seg_{un}} - \frac{P_{ref}}{Seg_{ref}} \right|
\]

where \( P_{un}, P_{ref} \) are the positions of an instance of a feature in an unknown word and a reference word respectively; \( Seg_{un} \) and \( Seg_{ref} \) are the number of vertical segments in an unknown word and a reference word respectively.

Taking into account the number of instances \( n \) of a feature; the fuzzy values \( FV_j \) and positional fuzzy value \( P_{-FV_j} \) of \( j^{th} \) instance, the confidence value \( CV \) of a feature in an unknown word with respect to a reference word is calculated as in equation (33).
In case the number of instances of a feature in an unknown word and reference word differ by 1, the nearest possible sequence match is taken to compute \( P_{\text{FV}} \) and if the difference is more than 1, confidence value of a feature is deemed as 0. The value of parameter 'd' is 1.0, if number of instances of a feature for unknown word and reference word are equal, else the value is lower than 1.

If number of instance of a feature is zero, \( P_{\text{FV}} \) cannot be computed using equation (32). CV of a feature is considered as 1, if number of instances of a feature is zero, both in unknown word and reference word. If number of instances of a feature is zero only for unknown word, CV is taken as 0.5 and if it is zero only for reference word, CV is taken as \((1.0 - \text{FV})\) where FV is the fuzzy value of a feature instance in an unknown word.

The confidence value of the first character is found out by comparing the first character with all reference characters using the method [261] of character dilation, centroid based normalization and distance computation. Closer the first character to the reference character, higher its confidence value.

3.4.4 Word and phrase confidence values

The matching word is found by comparing an unknown word representation with all the reference words representations in a word dictionary. The word confidence value \( CV_{\text{word}} \) of a reference word is computed as weighted mean of confidence values of 'm' features as given in equation (34) where \( w_i \) is the weight and \( CV_i \) is the confidence value of the \( i^{\text{th}} \) feature.
Weights of features depends upon the algorithms' efficacy of detecting them correctly and upon the magnitude of variation that is possible in features due to different style of writing. Weights for upper stroke and lower stroke features are taken as 1.0 while that of middle loop and first character are taken as 0.5. The word with the highest confidence value is considered as the match word. Word matching thus follows a bottom-up approach.

The phrase recognition makes use of contextual information following a top-down approach. Every phrase in a text dictionary has certain number of words and each word occupy a certain position in a phrase. An unknown phrase is partitioned into words by finding valid spaces between words. For each unknown word, first few matching words are found using equation (34). These matching words are probable candidates for the unknown word. Thus for each unknown word in a phrase, a set of probable candidates is found.

With dictionary look-up, the phrases with the number of words equal to number of words in unknown phrase are found. For each of these probable phrases, the confidence value is evaluated based on the confidence value of probable candidate for each unknown word. The phrase with the highest confidence value is considered as the matching phrase.

### 3.4.5 Words and text dictionaries

The text and word dictionaries are developed from predefined phrases. Text dictionary keeps contextual information of words in a phrase while the word dictionary stores holistic representation of words. For each phrase, the text dictionary stores the length of phrase in terms of characters, the phrase itself, number of words in phrase and the index of these words in a word dictionary.
The word dictionary stores the length of a word in terms of characters, the word itself, length in terms of vertical segments cleaving a word, count and position of upper strokes, middle loops and lower strokes. Number of word instances in predefined phrases, phrase indices containing the word and word position in phrase are also stored. Some portion of word and text dictionary is illustrated on page 126 and 127 respectively.

For each character 'A-Z' and 'a-z' information regarding number of vertical segments across a character and presence of upper stroke, middle loop and lower stroke are stored in a look-up table to assist construction of word dictionary. These information for capital and small characters is displayed in table 5 and table 6 respectively.

Thus given a phrase, its entries in text and word dictionaries gets automated. This also allows for dynamic expansion of dictionary and easy change over to a different set of phrases depending on the application’s domain.
Table 5: Features for capital characters A..Z

<table>
<thead>
<tr>
<th>Characters</th>
<th>Segments count</th>
<th>Upper strokes count</th>
<th>Middle loops count</th>
<th>Lower strokes count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>O</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6: Features for small characters a..z

<table>
<thead>
<tr>
<th>Characters</th>
<th>Segments count</th>
<th>Upper strokes count</th>
<th>Middle loops count</th>
<th>Lower strokes count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>q</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Text Dictionary (Truncated)

1 (19) [Advanced Computing] (2) [1,14]
2 (24) [Artificial Intelligence] (2) [6,36]
3 (18) [Basic Electronics] (2) [9,20]
4 (30) [Basic Electronics Engineering] (3) [9,20,22]
5 (30) [Chemical Reaction Engineering] (3) [11,58,22]
6 (22) [Computer Applications] (2) [13, 5]
7 (41) [Computer Aided Design Of Digital Systems] (6) [13, 2,17,49,18,64]
8 (18) [Computer Graphics] (2) [13,27]
9 (21) [Computer Programming] (2) [13,57]
10 (26) [Computer Programming In C] (4) [13,57,33,10]
11 (22) [Computer Organization] (2) [13,51]
12 (21) [Computer Peripherals] (2) [13,53]
13 (15) [Control Theory] (2) [15,66]
14 (32) [Data And Computer Communication] (4) [16, 4,13,12]
15 (24) [Data And File Structure] (4) [16, 4,25,63]
3.5 Modified Variation Measure using Average Entropy Difference

3.5.1 Introduction

Handwritten characters have large variation in shapes due to different writing styles. A measure is required to compute amount of variation in different data sets used by handwriting recognition algorithms.

A reference standard set of handwritten characters cannot be decided upon because of many styles of handwriting. Hence it is not possible to measure the variation in a character sample against a standard character.

The other approach to this problem is to take a group of same characters, that is the characters belonging to the same class. Say 500 samples of 'A', written by different persons. Now the variation in this data set can be computed by comparing these samples.

The research proposes a variation measure which modifies the Average entropy difference variation measure so that the measure becomes more stable against rotation of frequency distribution and withholds all the four properties: Boundedness, Independency, Monotonicity and Constancy mentioned in literature review, section 2.5. Apart from this, a variation measure should also be rotation invariant. For example, character such as '×' and '+' are similar except for rotation by 45 degree. If datum is consisting of several '×' or if it is consisting of several '+', the variation measure should not change just because of different inclination of a character type.

The variation value of data samples is computed using the proposed variation measure for two algorithms: one statistical and other structural algorithm.
3.5.2 The proposed variation measure

The Average Entropy Difference variation measure '\( V_d \)' given in equation (13) in section 2.5 satisfies the four properties of Boundedness, Independency, Monotonicity and Constancy and is reproduced here for convenience:

\[
V_d = \begin{cases} 
\frac{1}{|C|} \sum_{x=1}^{X} \sum_{y=1}^{Y} v(x, y) & |C| \neq 0 \\
0 & |C| = 0 
\end{cases} 
\] (35)

\[
v(x, y) = \frac{h(x, y)}{\alpha \cdot \max\{|h(x+1, y) - h(x, y)|, |h(x, y+1) - h(x, y)|\} + 1} 
\] (36)

where \( h(x, y) \) is average entropy, equation (11), \( C \) is the set of those points for which \( h(x, y) \) is non-zero and \( \alpha \) is a multiplying factor for range of \( v \).

The proposed variation measure '\( V_m \)' takes into account the 'rotation-invariant' property and is given as:

\[
V_m = \begin{cases} 
\frac{1}{|C|} \sum_{x=1}^{X} \sum_{y=1}^{Y} v_m(x, y) & |C| \neq 0 \\
0 & |C| = 0 
\end{cases} 
\] (37)

\[
v_m(x, y) = \frac{h(x, y)}{\alpha \cdot b(x, y) + 1} 
\] (38)

\[
b(x, y) = \max\{|h(x+1, y) - h(x, y)|, \frac{|h(x, y+1) - h(x, y)|}{\sqrt{2}}, \frac{|h(x+1, y+1) - h(x, y)|}{\sqrt{2}}\} 
\] (39)

where \( h(x, y) \) is average entropy, \( C \) is the set of those points for which \( h(x, y) \) is non-zero and \( \alpha \) is a multiplying factor for range of \( v_m \).

The entropy difference (the average entropy difference \( v \) in equation (36) and the modified entropy difference \( v_m \) in equation (38)) at a point depends directly on the entropy of that point and inversely on difference in entropy of neighboring points.
3.5.3 Case-based theoretical proof

A case based proof is given to prove that the proposed variation measure is rotation invariant to frequency distribution. Figure 21 shows four cases of neighboring points with different directions of maximum change in entropy. The lower left point \((x,y)\) in all the four cases has the entropy value 'i'. The change in entropy is considered to be \(\delta\) for unit distance.

Case 1 shows change in maximum entropy in \(x\)-direction, 0 degree, while case 3 show it in \(y\)-direction, 90 degree. Case 2 and case 3 shows change in maximum entropy at an inclination of 45 degree and 135 degree respectively.

Using equation (36) for the entropy difference \(v(x,y)\), case 1 and case 3 gives the value of \(v(x,y)\) as:

\[
v(x,y) = \frac{h(x,y)}{\alpha \delta + 1}
\]  (9)

Fig. 21: Direction of maximum change in entropy
while case 2 and case 4 gives the value of $v(x,y)$ as:

$$v(x, y) = \frac{h(x, y)}{\alpha \delta + 1}$$  \hspace{1cm} (10)

Using the modified values of variation measure given in equation (38) and equation (39), all the four cases gives the value of $v_m(x,y)$ as:

$$v_m(x, y) = \frac{h(x, y)}{\alpha \delta + 1}$$  \hspace{1cm} (11)

Equation (40) and equation (41) show that the value of entropy difference $v$ increases at 45 degree and 135 degree (case 2 and case 4) compared to value of $v$ at 0 degree and 90 degree (case 1 and case 3). The modified entropy difference $v_m$ remain same for all the four cases as given in equation (42) and it is same as of $v$ in equation (40). The discrepancy occurs because the entropy difference $v$ considers only the horizontal and vertical neighbors while in the proposed entropy difference $v_m$, the diagonal neighbors are also taken into account.

3.5.4 Properties

The proposed variation measure satisfies all the four properties of Boundedness, Independency, Monotonicity and Constancy which are mentioned in literature review. The proof of all these four properties is given below.

**Boundedness**

When the sample images are all same, the frequency distribution at each point in datum would have either value 0 (for all 0s) or M (for all 1s). Hence the probability distribution, equation (12), would be either 0 or 1. This would give the value of 'h' as 0 at each point and would make the set 'C' empty, and hence the value of $V_m$ would be 0, the 'lower' bound.

When the sample images are such that, the frequency distribution at each is (M/2),
the probability distribution would be \((1/2)\) at each point. This would give the value of 'h' as 1 at all points and would make 'v' to be 1. Hence \(V_m\) would turn out be 1, the 'upper' bound.

The lower limit and upper limit are thus fixed and are also independent of number of images (M) and size of image (S).

**Independency**

For all points which are white in all images or which are black in all images, the value of 'h' is zero, hence such points do not belong to set C. This makes \(V_m\) independent of paper size and character size.

**Monotonicity**

To prove the property of monotonicity, let us first understand the effect of linear expansion on the image data. Let a \(2 \times 2\) arbitrary portion of an image be represented as (i) in figure 22. The linear expansion in x-direction by a factor of 2 is shown in (ii) for image portion in (i), while the linear expansion in y-direction by a factor of 2 is shown in (iii) of figure 22.

![Fig. 22 : Linear expansion of \(2 \times 2\) portion of an image.](image)
In (ii), the variation value will remain same as in (i) for points at (2,1), (2,2), (4,1) and (4,2) as the neighboring points required by variation measure remain same as in (i). But for points at (1,1), (1,2), (3,1) and (3,2) the variation value would change compared to (i). Out of the three neighbors: horizontal, diagonal and vertical, the maximum difference will be obtained with the vertical neighbor. Let us call this variation value as $v_{mx}$, obtained due to linear expansion in x-direction, satisfies the following inequality.

$$v'_{mx}(x,y) = \frac{h(x,y)}{\alpha * (h(x+1,y) - h(x,y)) + 1} \geq v_m(x,y)$$  \hspace{1cm} (43)

In (iii), the variation value will remain same as in (i) for points at (1,2), (1,4), (2,2) and (2,4) as the neighboring points required by variation measure remain same as in (i). But for points at (1,1), (1,3), (2,1) and (2,3) the variation value will change compared to (i). Out of the three neighbors: horizontal, diagonal and vertical, the maximum difference will be obtained with the horizontal neighbor. Let us call this variation value as $v_{my}$, obtained due to linear expansion in y-direction, satisfies the following inequality.

$$v'_{my}(x,y) = \frac{h(x,y)}{\alpha * (h(x,y+1) - h(x,y)) + 1} \geq v_m(x,y)$$  \hspace{1cm} (44)

The formal proof can be given now. Consider the frequency distribution as \{f(x,y)\} and the modified average entropy difference as $V_m$. When \{f(x,y)\} is expanded linearly $k$-times along x-axis and $m$-times along y-axis, let the frequency distribution be denoted as \{f'(x,y)\} and modified average entropy difference as $V'_m$.

For each $i,j$,

$$f(p,q) = f(i,j)$$

for $p = k(i-1)+1, k(i-1)+2, \ldots, k_i$ and $q = m(j-1)+1, m(j-1)+2, \ldots, m_j$.

$$V'_m = \frac{1}{C} \sum_{(x,y) \in c} v'_m(x,y)$$

$$= \frac{1}{km|C|} \left[ \sum_{(x,y) \in c} v_m(x,y) + (k-1) * \sum_{(x,y) \in c} v_{mx}'(x,y) + (m-1) * \sum_{(x,y) \in c} v_{my}'(x,y) + \right.\left. \right.$$
\[
(k-1) (m-1) \sum_{(x,y) \in c} h(x,y) \geq \frac{1}{km|c|} \left[ \sum_{(x,y) \in c} v'_m(x,y) + (k-1) \sum_{(x,y) \in c} v_m(x,y) + (m-1) \sum_{(x,y) \in c} v_m(x,y) + (k-1) (m-1) \sum_{(x,y) \in c} v(x,y) \right] = V_m
\]

This proves that the proposed variation measure \( V_m \) increases monotonically when the frequency distribution is expanded linearly.

**Constancy**

Consider the frequency distribution as \( \{f(x,y)\} \) and the proposed average entropy difference as \( V_m \). When \( \{f(x,y)\} \) is copied \( k \)-times along \( x \)-axis, let the frequency distribution be denoted as \( \{f'(x,y)\} \) and modified average entropy difference as \( V'_m \). Hence,

\[
f(i+pX,j) = f(i,j) \quad \text{for } p = 1, 2, \ldots, k.
\]

\[
v'_m(x,y) = \cdots = v'_m(x+(k-1)X,y).\]

\[
|C'| = k |C|.
\]

Now,

\[
V'_m = \frac{1}{|C'|} \sum_{(x,y) \in c'} v'_m(x,y)
\]

\[
= \frac{1}{k|C|} \left[ k \sum_{(x,y) \in c} v'_m(x,y) \right]
\]

\[
= \frac{1}{|C|} \sum_{(x,y) \in c} v'_m(x,y)
\]

\[
= V_m
\]

The above proof shows that the modified variation measure follows all the four properties mentioned in the literature, required by handwritten character variation measure.
3.5.5 Variation value of real data samples

The variation value of samples belonging to the same class can be computed using equation of variation measure, provided all sample images have same height and width. But in real data samples, this is not the case.

Hence, before computing the variation value, the size of all images is to be made equal. This can be done by considering the maximum value of width and height from given image samples, and then zooming all sample image to this maximum size.

Property of Monotonicity states that increase in gray area of datum, increase the variation value monotonically. Zooming can be done by replication method which will increase the variation value. This is desired as different 'size' of samples, do indicate variation.

The steps to find variation value of real data are given in the flowchart on page 136. Here the maximum size of image along with the frequency distribution of image data is computed as each data samples is being considered. After evaluating frequency distribution of datum, the probability distribution and variance value are computed.
Flowchart for computing variation value of real data

Start

Number of images = 0

Read image data from a file as current data

Find size of image as current size

Is it first file?

Y

Find maximum size from current size & previous size

Expand previous image data and current image data to maximum size

Compute frequency distribution

Increment number of Images

Let current data be previous data

Let current size be previous size

N

Are all files over?

Y

Compute probability distribution

Compute variance value

End