

## 2. Development of OHWR System for Telugu Language

V. S. Chakravarthy & Team  
IIT, Madras

### 1. Online Recognition of Characters Using SVM

A system for online recognition of Telugu characters in which strokes are recognized using Support Vector Machines and the character is recognized based on the trained main rules using SVM.

#### 1.1 Algorithm

- ✦ Pre-processing
- ✦ Normalization
- ✦ Feature Extraction
- ✦ Vowel and Consonant Stroke Recognition
- ✦ Main Stroke Recognition
- ✦ Pre-classification
- ✦ Auxiliary Strokes (Top, Bottom, Base) Recognition
- ✦ Character Recognition

##### 1.1.1 Pre-processing

Pre-processing the data transforms the stroke to a uniform representation irrespective of variations within examples of the same class. It reduces some of the variability of handwritten data. A statistical classifier like a SVM imposes a constraint that feature vectors should have a fixed length representation. For this purpose, the stroke is interpolated to a

fixed number of data points,  $N(64 \text{ here})$ , which is determined empirically.

#### 1.1.2 Normalization

Normalization is a step that is helpful to remove variations that occur during writing and to obtain a uniform representation of the Strokes.

Steps involved in Normalization are

##### i) Translation

Every stroke has to be located uniformly with respect to the origin to remove the translational variation. This is achieved by the following procedure; take the leftmost and the bottom most point of the stroke. Subtract the leftmost point from the x coordinates and subtract the bottom most point from the y coordinates of the stroke. This step provides a uniform representation of the stroke.

##### ii) Scaling.

The size of a handwritten character varies with writers and for different instances of the same character for a particular writer. For uniformity in representation, it is necessary that strokes are normalized in size. The approach used for size normalization is scaling based on the height and width of the stroke. To get the uniform representation of the stroke we are dividing x coordinates by stroke width and y coordinates by stroke height.

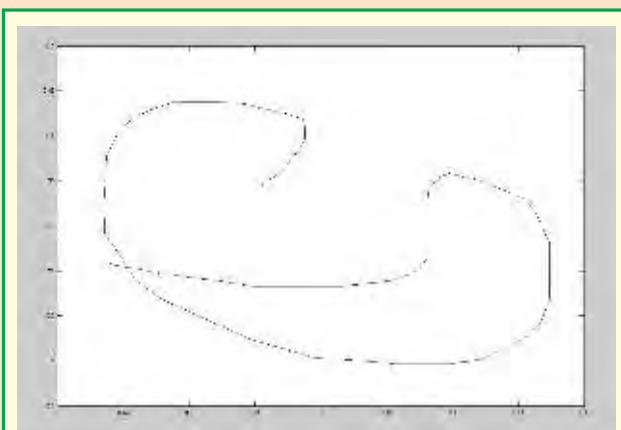


Fig 1.1 Main Stroke before normalization

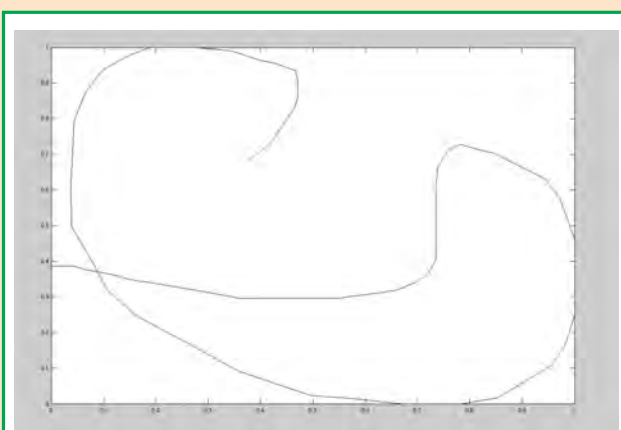


Fig 1.2 Main Stroke after normalization

The result of normalization is a stroke whose leftmost and bottommost stroke point is at (0, 0) and the height and width of the stroke is 1.

### 1.1.3 Feature Extraction

The goal of feature extraction is to identify unique traits that are not susceptible to distortion in each stroke. Feature Extraction that is used here is FFT (Fast Fourier Transform). Handwriting is viewed as the result of an oscillation process, with each axis representing a time varying signal. Spectral features represent the stroke in the frequency domain. The discrete Fourier transform (DFT) operation is used to transform the handwriting signal from the time domain to the frequency

domain. The stroke is sampled to a fixed number of points and a 1-dimensional array from the abscissa and ordinates is created called feature vector. Since the handwriting is a low-pass signal, few frequency components need to be considered. The choice of the number of coefficients to be considered has been determined empirically.

### 1.1.4 Vowel and Consonant Stroke Recognition

To enhance the main stroke classification accuracy we have split the main stroke into vowel stroke and consonant stroke. The Fourier features represented by the sequence of positional co-ordinates in pre-processed main stroke is passed to the vowel SVM classifier and the Consonant SVM classifier.

The result is a vector with a value indicating relative strength of each class for the corresponding vowels and the consonants respectively.

### 1.1.5 Main Stroke Recognition

The stroke that is first written for a character is known as main stroke. The output vectors from the Consonant and Vowel SVM clubbed with raw points of the first stroke are passed to the main stroke recognition SVM. The output is a 149 element vector with values indicating relative strength of each of the 149 main strokes. Here, the vector entry with largest value received from the main stroke SVM is compared with the existing main stroke labels and the main stroke is recognized based on the matching stroke label or id.

### 1.1.6 Pre-classification

Preclassification seeks to classify the strokes in a character into three categories: 1) base stroke 2) top stroke 3) bottom stroke. The

first stroke in the character is considered main stroke and the rest of the strokes are used in pre-classification to be classified into one of the three classes considered together as auxiliary strokes.

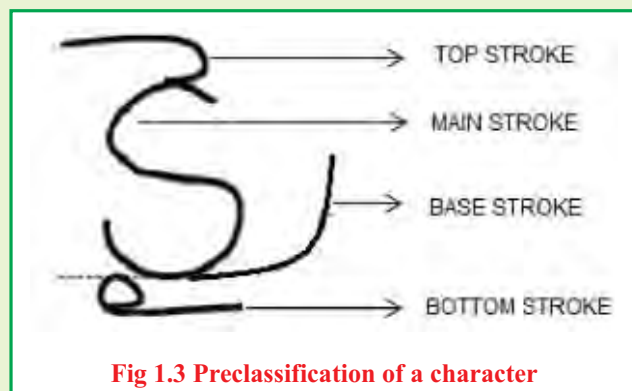


Fig 1.3 Preclassification of a character

- ✦ The base stroke is identified by a histogram of the y-co-ordinates of the stroke. It is the stroke closest to the interval with the least number of points.
- ✦ The top and bottom strokes are identified by considering various spatial features relative to the baseline stroke, including the distance of the constituent strokes from x axis and y axis, the normalized Euclidean distance between the baseline strokes and the other constituent strokes, height above and below the baseline, location as defined by the octant in which the strokes are present.

To recognize the auxiliary strokes, the relative position of the auxiliary stroke is to be preserved. For this purpose, main stroke height based normalization of the main stroke and corresponding auxiliary stroke to be classified is done. The normalized stroke points from the main and auxiliary stroke are interpolated to 32 points each. A feature vector with 32 points from the main stroke and 32 points from the corresponding auxiliary stroke is passed to the pre-classification SVM.

The result of the pre-classification SVM is 3 element vector corresponding to the 3 classes of auxiliary strokes, the maximum of which gives the index to the class as follows.

- ✦ Class 1 – The stroke is a base stroke.
- ✦ Class 2 – The stroke is a bottom stroke.
- ✦ Class 3 – The stroke is a top stroke.

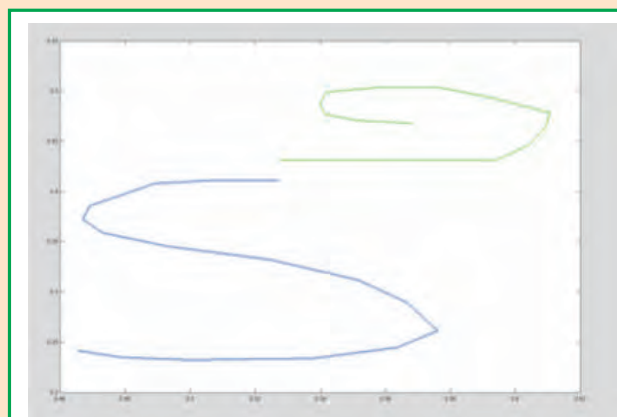


Fig 1.4 Character before main stroke height based normalization

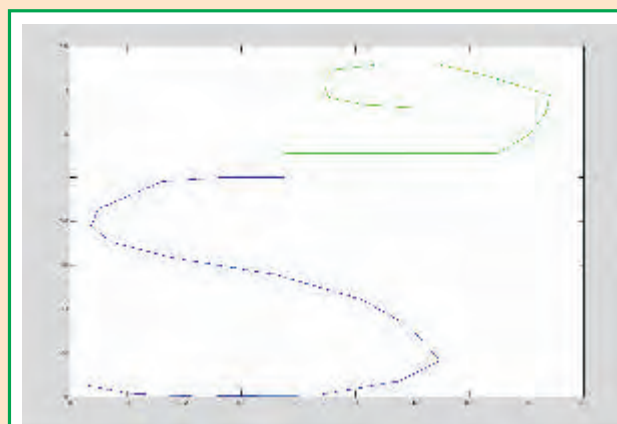


Fig 1.5 Character after main stroke height based normalization

### 1.1.7 Top, Bottom, Base Strokes Recognition

Different classifiers are built for the top, bottom and base strokes. Based on the stroke class, the stroke is passed to the corresponding SVM. The auxiliary stroke raw points are unit square normalized, interpolated to 64 points, fft of the 64 points is obtained and the 128 fft (64 real and 64 imaginary) coefficients are

passed as a feature vector to the corresponding SVM as below.

- ✦ Class 1 –The feature vector is sent to the Base SVM with 20 classes.
- ✦ Class 2 –The feature vector is sent to the Bottom SVM with 36 classes out of which only 24 are considered after ignoring the C2 stroke classes.
- ✦ Class 3 –The feature vector is sent to the Top SVM with 22 classes.

### 1.1.8 Character Recognition

This is the last step in handwriting recognition. Here the feature vector received

is of size 1x215(215 strokes-149 main strokes from index 1 to 149, 20 base strokes from index 150 to 169, 24 bottom strokes from index 170 to 193, 22 top strokes from index 194 to 215) and is passed to the character SVM engine. The vector entries with largest value corresponding to main stroke and 3 auxiliary strokes are compared with existing stroke labels and the character is recognized as a combination of the stroke labels. The output is a 527 element vector with values indicating relative strength of each of the 527 characters. The index of the vector entry with the largest value is used to determine the character.

### 1.1.9 Results

	SVM	Data used for classification	Data Classified Correctly	Efficiency (%)
1	Main stroke	20654	20303	98.29
2	Pre-classification	18572	18506	99.64
3	Base stroke	3678	3528	95.92
4	Bottom stroke	7487	6927	92.52
5	Top Stroke	7407	7216	97.42
6	Char	14146	13643	96.44

Table 1.1 SVM classification results.

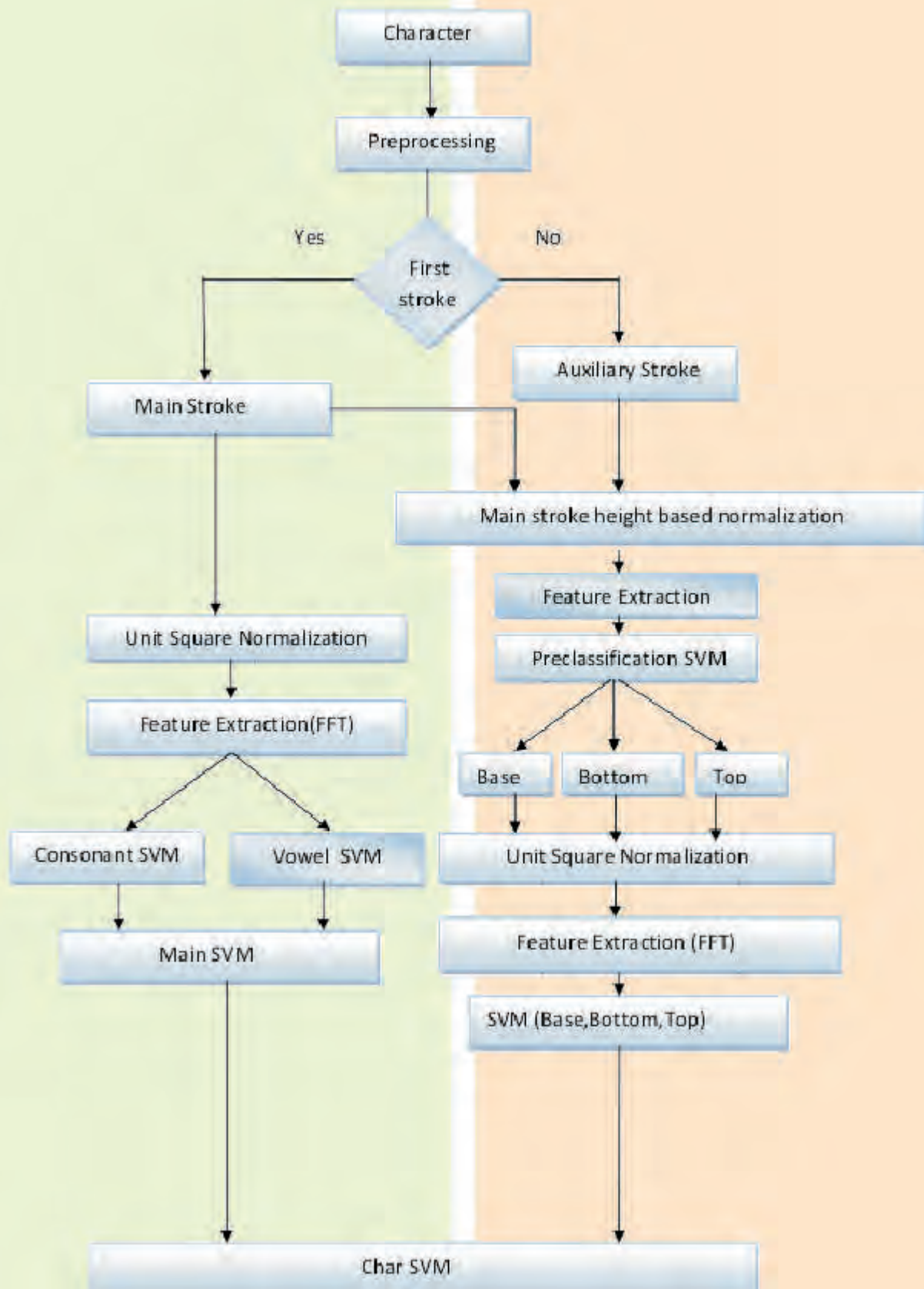


Fig 1.6 Flowchart of SVM Recognition of characters



## 2. Offline Recognition of Characters

Offline character recognition refers to the process of recognizing characters stored digitally. It is conventional to perform further processing to achieve higher accuracy in recognition. Offline recognition of characters is done using Convolutional Neural Networks (CNN) and Autoencoders.

### 2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are multilayer networks which learn complex, high-dimensional, non-linear mappings from large sets of data samples.

#### 2.1.1 Introduction

In conventional models of pattern recognition, a feature extractor extracts the feature vectors and a classifier categorizes these feature vectors into classes. A more advanced method eliminates the feature extractor and directly feeds the network with normalized images. The network layers are trained by supervised or unsupervised learning and extracts the appropriate features

The input layer is fed with (binary 2D) images that are normalized. Each unit of any subsequent layers receives input from a set of units located in the small neighbourhood in the previous layer (Local receptive fields). With these receptive fields, neurons extract basic visual features such as oriented edges, end points, corners. These basic features are then combined by the higher layers in the recognition process. Distortions in the input may cause the position of features to vary and

also basic feature detectors that are useful on one part of the image are likely to be useful across entire image. So, a set of neurons whose receptive fields are located at different places on the image are forced to have identical weight vectors. The output of all such neurons together is called a feature map. A feature map is a plane of neurons that shares a single weight vector. Units in a feature map perform the same operation across entire image.

Each Convolutional hidden layer is composed of several feature maps so that multiple features can be extracted from each location. In above figure, the first hidden layer contains four feature maps with 5 by 5 receptive fields. After a feature has been detected, its exact location is of less importance as long as its relative position with respect to other features is known. So, after each Convolutional layer, a sub sampling layer is designed which performs local averaging and sub sampling, thus reducing the resolution of each feature map. This also helps in reducing the sensitivity of output to shifts and distortions. In the above figure second hidden layer performs 2 by 2 averaging and sub sampling. After each sub sampling layer, number of feature maps is increased as the spatial resolution is decreased. After the sub sampling layer, each unit in the next Convolutional layer can have input connection from several feature maps in the previous layer. i.e., new features are extracted by combining several existing features.

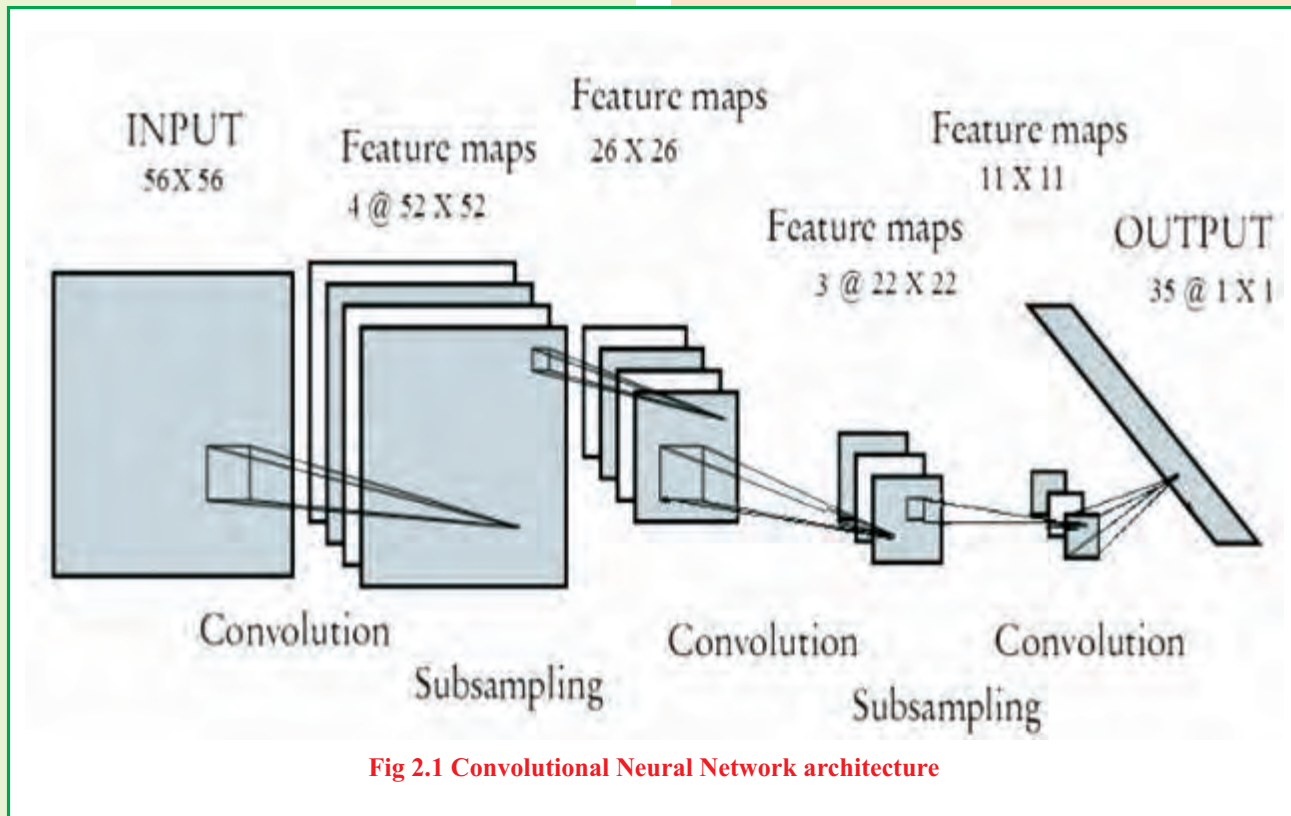


Fig 2.1 Convolutional Neural Network architecture

## 2.2 Training The CNN: Supervised Learning Using Back-propagation

Back propagation is a supervised method which is an implementation of the Delta rule and requires a teacher that can calculate the desired output for any given input. In CNNs, weights can be learned through back propagation i.e. it can be viewed as CNNs synthesize their own feature vectors. The weight sharing technique reduces the number of free parameters and also improves the generalization ability of the network.

## 2.3 Telugu Character Recognition Using CNNs

Online stroke information of telugu hand written characters are normalized with leftmost and bottommost point at (0, 0) and main stroke height 1 and then converted into binary images of size 56 x 56/28 x 28 pixels (2D images). These images are input to the CNN. The index of the detected character is then used to display the character.

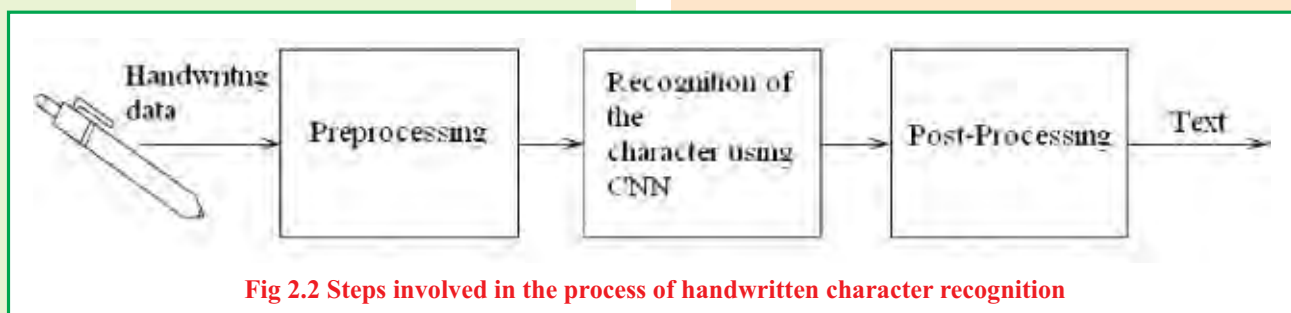


Fig 2.2 Steps involved in the process of handwritten character recognition

Telugu script consists of 16 vowels and 36 consonants. However, there are composite characters of CV type (combination of a vowel and a consonant) about 576 (36x16) and of CCV type (combination of 2 consonants and a vowel) about 20736(36x36x16). For the CNN network we consider only the characters of C, V and CV type i.e.628 (576 + 16+36) to be classified.

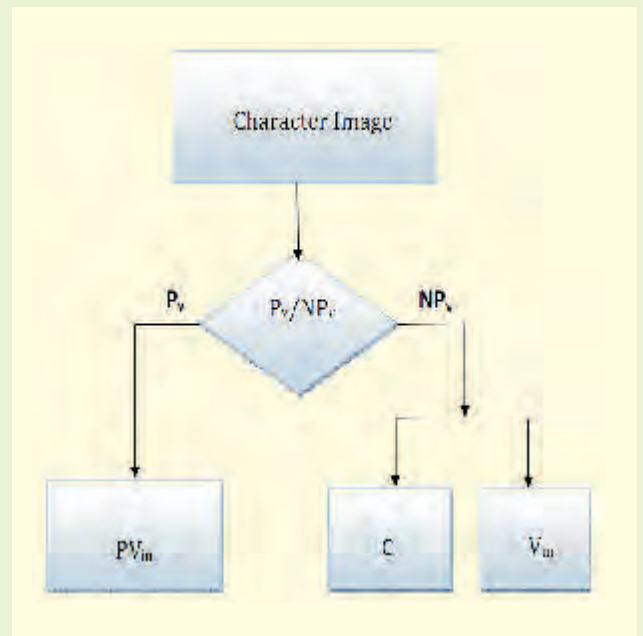
A Network has to be developed to recognize all C, V, CV type characters. The network for recognizing all 628 CV characters must have 628 classes in its output layer. This makes the network complicated and training this network is time consuming and may result in poor performance. In order to minimize the complexity of the network, instead of having a single network to recognize the entire set of C, V, CV characters, we have four separate networks

- ✦ Classify the character as a vowel or a consonant – 1 network (PV)
- ✦ If Vowel
  - o Recognize the vowel – 1 network (PVin)

✦ If Consonant

- o Recognize the consonant – 1 network (C)
- o Recognize the vowel modifier for the consonant – 1 network (Vm)

Thus the number of classes in the output layer reduces to 2 for PV, 16 for PVin, 36 for C and 15 for Vm networks respectively.



**Fig 2.3 Separate Networks in CNN**

S No	Layer name	No. of feature Maps	Size of feature maps	Total no. of Neurons
1	Input layer	1	56x56	3136
2	First hidden layer	4	52x52	2704
3	Second hidden layer	4	26x26	676
4	Third hidden layer	3	22x22	484
5	Fourth hidden layer	3	11x11	121
6	Output layer	1	16	16

**Table 2.1: Specifications of CNN for Vowel recognition : (PVin)**



S No	Layer name	No. of feature Maps	Size of feature maps	Total no. of Neurons
1	Input layer	1	56x56	3136
2	First hidden layer	4	52x52	2704
3	Second hidden layer	4	26x26	676
4	Third hidden layer	6	22x22	484
5	Fourth hidden layer	6	11x11	121
6	Output layer	1	36	36

Table 2.2: Specifications of CNN used for Consonant recognition : (C)

S No	Layer name	No. of feature Maps	Size of feature maps	Total no. of Neurons
1	Input layer	1	56x56	3136
2	First hidden layer	4	52x52	2704
3	Second hidden layer	4	26x26	676
4	Third hidden layer	3	22x22	484
5	Fourth hidden layer	3	11x11	121
6	Output layer	1	2	2

Table 2.3: Specifications of CNN used for pure vowel recognition : (PV)

S No	Layer name	No. of feature maps	Size of feature maps	Total no. of Neurons
1	Input layer	1	56x56	3136
2	First hidden layer	4	52x52	2704
3	Second hidden layer	4	26x26	676
4	Third hidden layer	3	22x22	484
5	Fourth hidden layer	3	11x11	121
6	Output layer	1	15	15

Table 2.4: Specifications of CNN used for Vowel modifier recognition : (Vm)

### 2.3.1 Results for CNN using Back Propagation

#### 2.3.1.1 Multistroke Characters

Multistroke characters are to be trained separately first and then this set of multistroke characters are added to the trained single stroke characters and training is done with the combined data set (Single stroke & multi stroke characters).

In order to test the network from the

available set of multi stroke CV characters a total of 25 different characters are collected from the hand writings of 10 different persons. This data set of 250 images (25x10) is trained over both the networks (consonant and vowel). At every stage of 50 runs the accuracy (% of the characters recognized) of the training set data is checked for both the networks. The below figure shows the graphs plotted for both the networks.

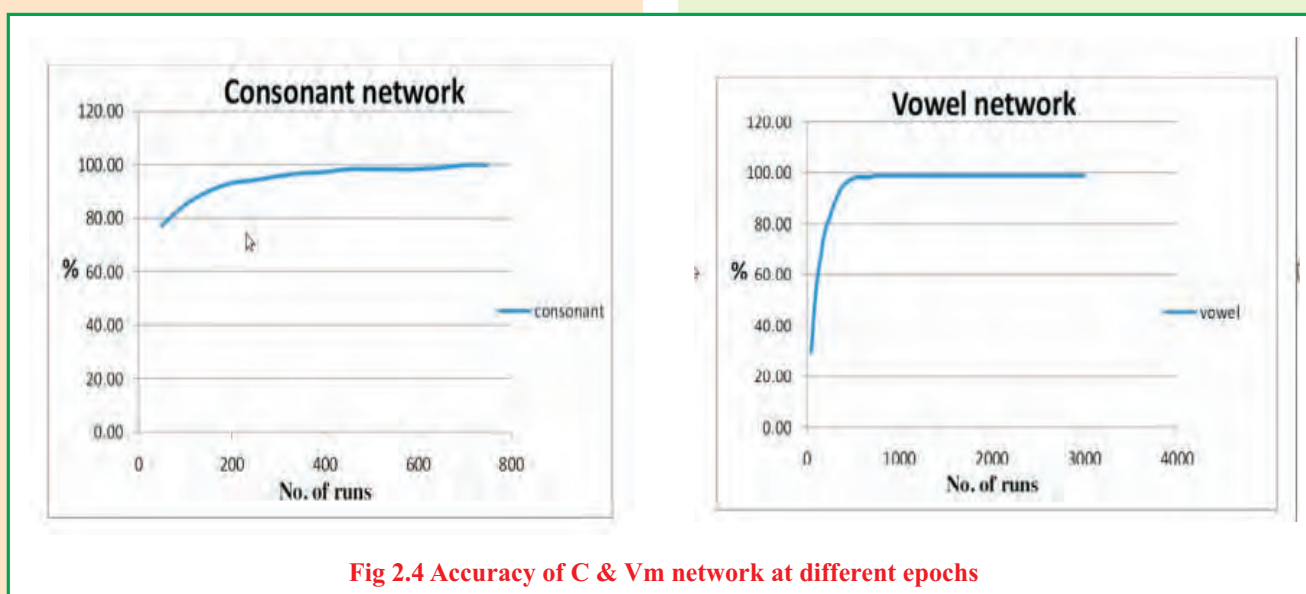


Fig 2.4 Accuracy of C & Vm network at different epochs

S.No	Network	Runs	Efficiency
1	Consonant	700	100
2	Vowel	750	99

Table 2.5: Optimum accuracies of the consonant and vowel networks for MS characters with 256 images:

The training is continued until the accuracy (% of the characters recognized) of the network reaches to an optimum value beyond which there is no further increase (weights of the network are adjusted in a way that the no. of characters recognized are maximum).

After yielding the optimum value

(accuracy) for this set of data, the network is tested with sets of the selected 25 multi stroke characters written by different persons, which are not included earlier in the training set. This is to see how far the network is able to learn in recognizing the characters (as the testing is done with data set which is not included in the training set).

S.No	Testing data set	Consonant accuracy	Vowel accuracy
1	R1	71	48
2	R2	90	50
3	R3	90	48
4	R4	95	55
5	R5	82	55
6	R6	86	76

**Table 2.6: Accuracies of the consonant and vowel networks**

Comparing the accuracies yielded by the testing data reveals whether or not the network is able to recognize the characters. If the accuracy (% of characters recognized by the network) for the testing data is not 100%, then the network is not trained with sufficient data. So the network has to be trained further with more of these selected MS characters of different hand writings.

### **2.3.1.2 Combined Stroke (CS) characters (single and multistroke)**

The networks are individually trained with the single and multi stroke characters and each has its own % accuracy for the testing data (untrained characters) which is quite promising and the % accuracy can be increased if more data is added.

However, we need to develop a single network (consonant and vowel) for the entire set of characters (single and multi stroke). To make that happen, first we need to train the network optimally with one set of characters (say single stroke) and then another set of

characters (multistroke) is added to the earlier set of characters (single stroke), and the entire set is trained together. Then the network is tested with untrained data to analyze the extent of ability of the network in recognizing the characters.

Single stroke characters with set of 4970 (70x71) images of 70 different characters collected from 71 different people is already trained with the consonant network with testing accuracy (80-85%) and vowel network with testing accuracy (80-90%). Multi stroke characters with a set of 250 images (25x10) of 25 different characters collected from 10 different people is added and trained with respective networks. Now the network is trained with 5220 (4970 SS +250 MS) images. At every stage of 50 runs the accuracy (% of the characters recognized) of the training set data is checked for both the networks. The network is trained until the accuracy has reached an optimum value beyond which there is no further increase.

S. No	Network	Runs	Accuracy
1	Consonant	2600	98
2	Vowel	3000	97

**Table 2.7: Optimum accuracies of the consonant and vowel networks for CS characters with 5220 (4970 SS+250 MS) images**

### 2.3.1.3 Fixed Position (Origin)

The concept of continuous training yielded a reasonably good (testing) accuracy and for achieving better accuracy more data must be used for training. Concentrating on certain aspects may enhance the recognition with minimal data. One such aspect is the positioning of the character in a given image. The character might be at any extreme corner of the given image.

Any character is normalized according to the written size and fitted in to the 56x56 binary pixel images. There is no fixed point in the 56x56 binary images where a character gets positioned. If the character gets positioned at any extreme the recognition by the network becomes difficult. In order to overcome this difficulty an adjustment procedure is proposed. This procedure is named the concept of fixing position. In this

procedure, the position of the character in the 56x56 input images is adjusted for the network to have better chance of recognition. This adjustment might help the network in developing a strategy of its own (as the position of the character is fixed at a certain point) in recognizing the character and hence in improving the overall testing accuracy.

The origin is defined as the intersection of two axis one horizontal and one vertical. The horizontal line is called the base line which is 19 pixels from the bottom end. The vertical line splits the image into two halves. The reason for shifting the base line is to segregate the main stroke of the character from the other strokes. The space below the base line accommodates the vowel part in case of CV and the C2 (consonant 2) of the CCV characters. So the origin is fixed.

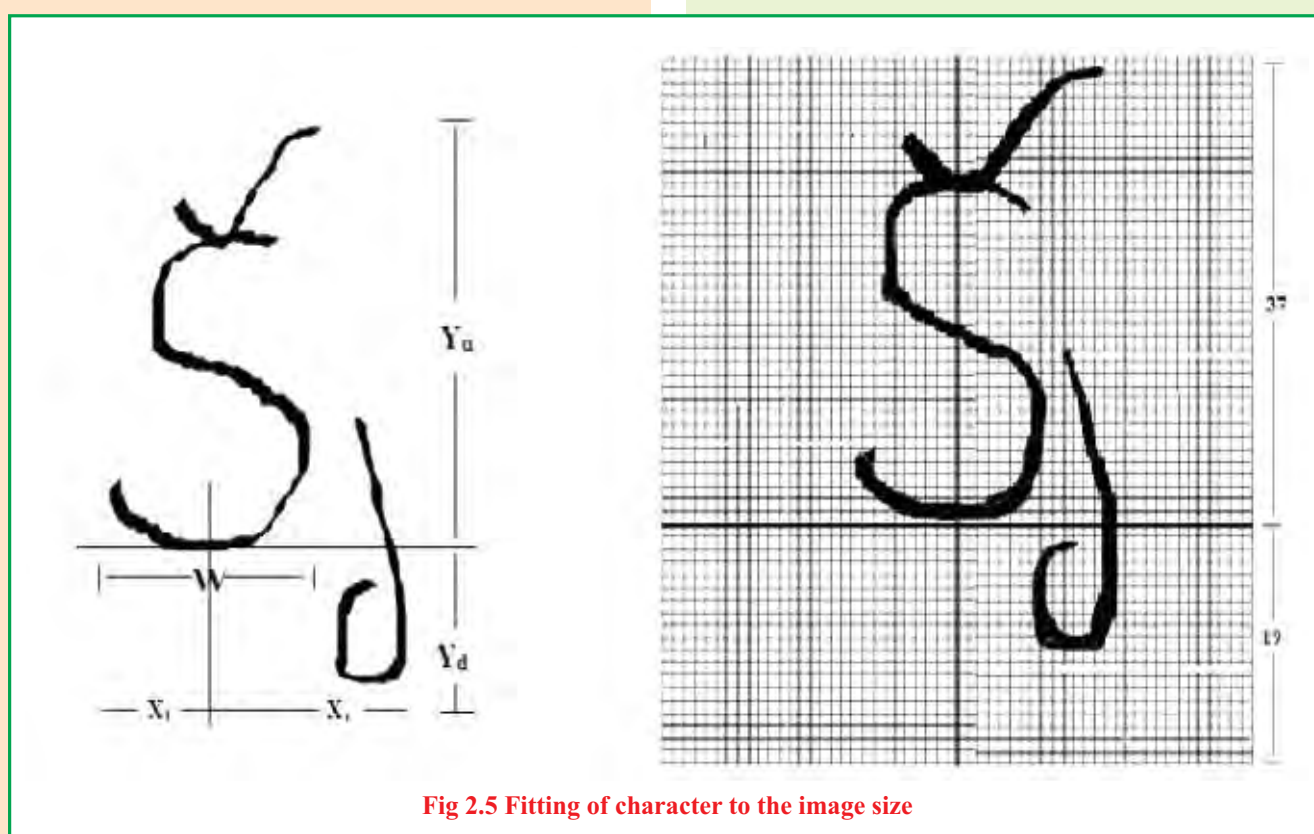


Fig 2.5 Fitting of character to the image size

The character has to be fit in the 56x56 image with the origins coinciding each other (Ao & Vo) and the dimensions of the character are adjusted with each multiplied by  $\lambda$ , such that the entire character fits into the image with the maximum length of the character fitting exactly into the provided lengths of the 56x56 image from the origin Vo without any loss of the character (with the maximum length fitting perfectly makes entire character fit).

Thus each and every character, whether it be single stroke or multi stroke, is adjusted into the 56x56 pixel image. It looks as if each character is started with the Vo as origin and spread over the 56x56 pixel images. These adjusted images of the characters are given as

input to the CNN for training. This concept definitely makes it easier for the network in developing the strategy of recognizing the characters relating the positions of the characters which start from a fixed position and improves testing accuracy of the untrained data.

Till now the networks are trained with images which vary in position even for the same character. A set of 26 MS characters collected from hand writings of 10 different persons is used for training. The data with 260 images is trained simultaneously with both the networks (consonant & vowel) one without fixing the position (base line) and another fixing the position.

S.No	Network	Runs		Accuracy	
1	Consonant	1300	1100	100	99
2	Vowel	4300	4000	97	95

Table 2.8: Optimum accuracies yielded by the consonant and vowel networks for MS characters with 520 images

S.No	Testing data set	Consonant Accuracy		Vowel Accuracy	
		With	without	with	Without
1	R1	77	58	65	35
2	R2	92	77	58	38
3	R3	89	82	64	35
4	R4	82	79	75	32

Table 2.9 : Testing accuracies of both the networks for MS (untrained) characters with and without fixing position

Network	Training			Testing	
	Data	Epochs	Efficiency (%)	Data	Efficiency (%)
PV/NPV	9448	13000	98.2	179	96.08
PVin	726	15500	98.95	478	85.56

Table 2.10 : Testing accuracies for MS characters for pure vowel (PV) and vowel index (PVin) networks.



## 2.4 Recognition Using PCA\_CNN-SVM

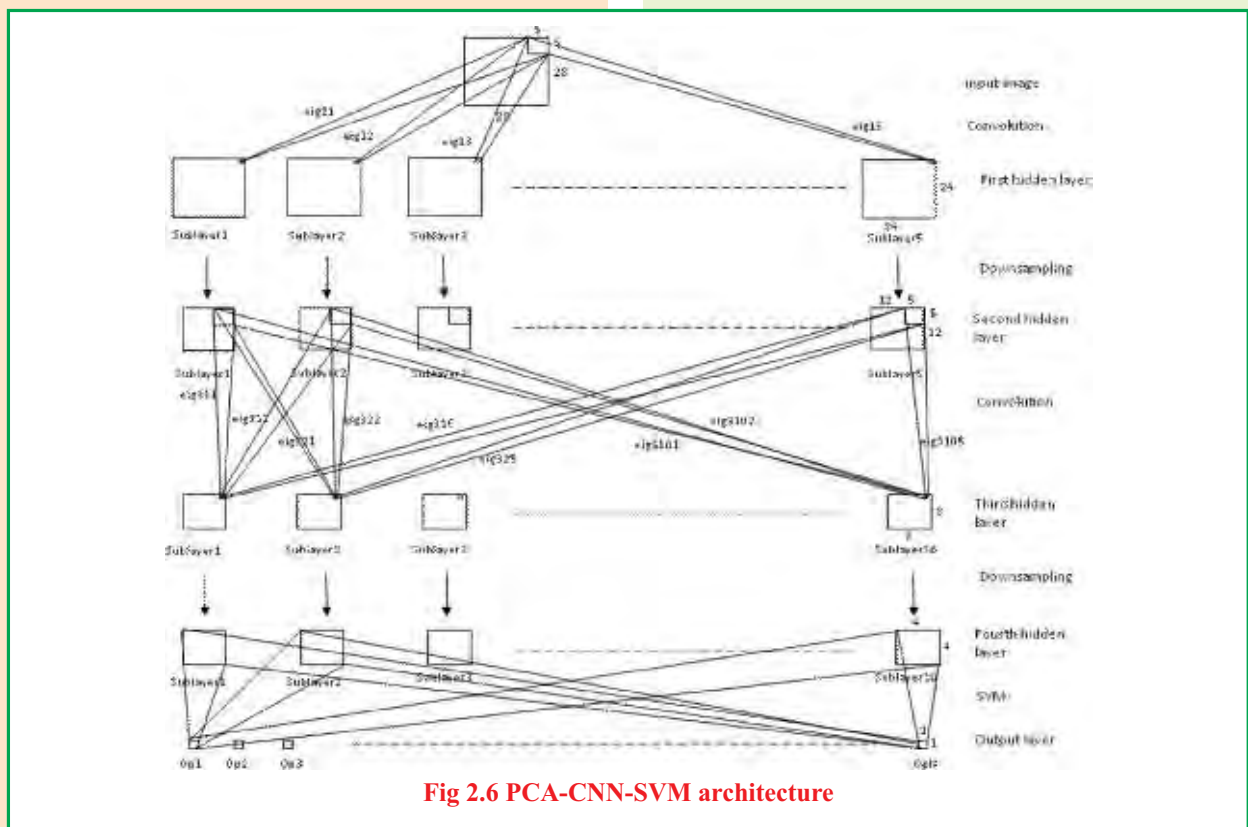
The Convolutional Neural Network initialised with random weights and trained using back propagation algorithm gave classification accuracies of 99.4413 during training and 95.1117 during testing for PV network and 100 for Pvin networks. For C and Vm networks the classification accuracy reached a maximum value of 46.97% and 80.99% respectively. To further improve the performance of C and Vm network, alternate approaches for training CNN are proposed.

### 2.4.1 Algorithm

#### 2.4.1.1: Unsupervised training of CNN hidden layers

A CNN with four hidden layers is chosen as the model for individual classifiers as in Fig. 2.6. The first and third weight stages are feature mapping stages which are trainable, while the second and fourth stages are non-

trainable subsampling stages. Autoencoders have been used in the past to train the feature mapping stages of CNN. An unsupervised training method like the autoencoder provides an excellent weight initialization and greatly reduces training time. A single hidden layer autoencoder network is an indirect way of performing Principal Component Analysis (PCA). Therefore, instead of using the slow backpropagation algorithm for training the autoencoder network, use of fast matrix-inversion based methods for PCA to train CNN weights has its advantages. Hence, we propose to use PCA to train the first (H1) and third (H2) (feature mapping) weight stages of CNN in that order. For each neuron in a given feature mapping layer, the responses of neurons in its receptive field are collected over the training set, and PCA performed over the set of responses.



The CNN network uses unsupervised weights for the first and third hidden layers that are obtained using PCA. The process for obtaining weights using PCA is as follows. A large set of images of size 28x28(11 x 11) are taken as training set. From each image in the training images, subimages of receptive field size 5x5(r x r) are obtained at the same corresponding location and these subimages are used for obtaining the Principal Components as in Fig. 2.7. Most n significant principal components are taken and are used

as weights to the corresponding neuron in the n sublayers of first hidden layer,

$$wl_{ij}^k = ql_{ij}^k \quad (1)$$

where,

$wl_{ij}^k$  = Weight vector of the neuron at location (i,j) position in the k'th sublayer of first hidden layer

$ql_{ij}^k$  = k'th principal eigenvector of the subimages that project to neurons at locations (i,j) in  $k^{th}$  sublayer of the first hidden layer.

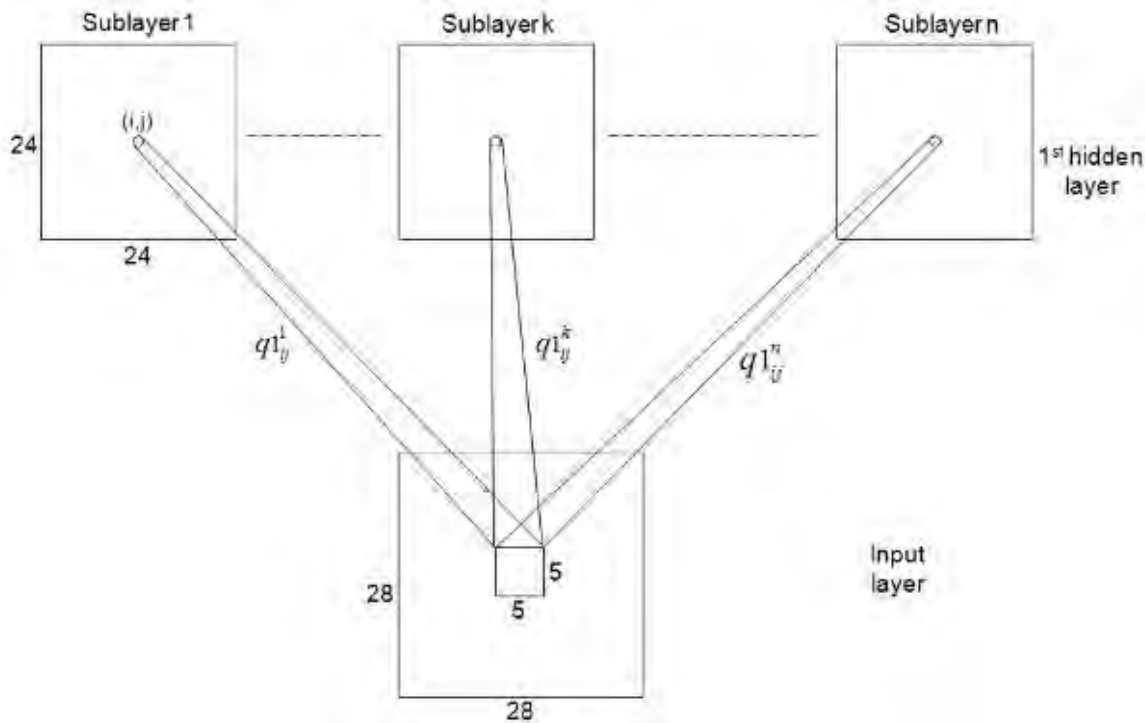


Fig 2.7 Determining weights during training for each neuron in the first hidden layer

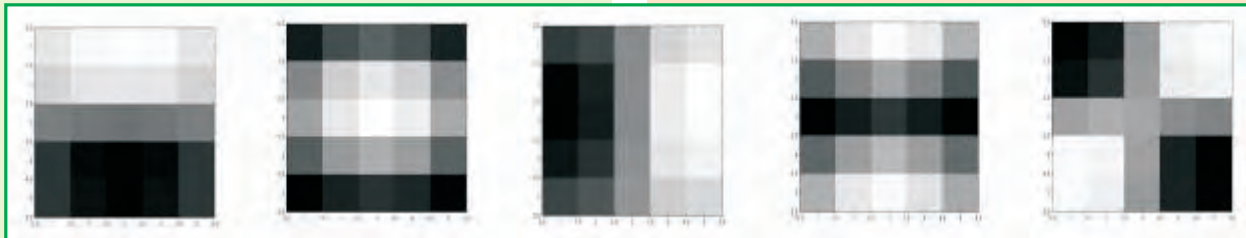


Fig 2.8 Weights from input layer to the five sub layers of first hidden layer for the first neuron with receptive field 5x5.

The same process of obtaining subimages and principal components is repeated to obtain the weights from the second hidden layer sublayers to third hidden layer sublayers as in Fig. 3 with the difference that “n” sublayers of size 12x12(13x13) are mapping onto “m” sublayers of size 8x8(14x14). The response of the network till the third hidden layer for all the training images is obtained using the weights calculated for first hidden layer above. Subimages of receptive field size 5x5(r x r) are obtained at the same corresponding location for each of the “n” sublayers of the second hidden layer and are concatenated to produce a r\*r\*n-dim vector. This r\*r\*n-dim response vector projects to a unique neuron in each sublayer of the third hidden layer. PCA performed over the entire stack of r\*r\*n-dim response vectors from a given location in the second hidden layer, is used to set the weights of the neurons which the r\*r\*n-dim response vector projects to. In other words,

$$w3_{ij}^k = q3_{ij}^k \quad (2)$$

where,

$w3_{ij}^k$  = r\*r\*n-dim weight vector of the neuron at location (i,j) position in the k'th sublayer of third hidden layer

$q3_{ij}^k$  = r\*r\*n-dim k'th principal eigenvector that projects to neuron at location (i,j) in kth sublayer of third hidden layer.

#### 2.4.1.2: Supervised training of the last stage using Support Vector Machines

The weight stage from the last hidden layer of the CNN to the output layer is trained using SVM. The entire hierarchy of CNN layers from the input to the last hidden layer may then be considered as a kernel layer of the SVM. Training the last, linear weight stage with SVM instead of backpropagation, which

actually reduces to delta rule for the last weight stage, is expected to improve generalization and reduce training time. The response of the fourth hidden layer, of size 4x4 (15 x 15), with “m” sublayers, which constitutes a 15\*15\*m-dim feature vector, is the input to the SVM, which is trained with a gaussian kernel, using SVM Torch. A Gaussian kernel SVM is used for classification, with an empirically determined value of width ( $\sigma$ ) and tradeoff (C) parameter. The value of  $\sigma$  used is 1 and the value of C is chosen as 100. The support vectors obtained during training are used to determine the output class.

#### 2.4.1.3. Ensemble Classifier

To further improve the performance accuracy of the method, instead of a single classifier an ensemble of classifiers is used. An ensemble classifier is obtained by combining multiple classifiers with least correlation in the performance on the validation set. 9 different networks of different sizes have been trained on the training set. The networks are ranked on the basis of their performance on the validation set. Two networks among these 9, with maximum performance and least correlation amongst them, were selected. The output of the ensemble network is decided by using 2 approaches. In voting method #1, the output vectors of the 2 networks are added and the class with maximum value was picked as the output class. In voting method #2, the class with the maximum value amongst the two networks was assigned as the output class. The ensemble that gave the best performance over the validation set was chosen. The ensemble classifier gave better performance accuracy compared to the individual classifiers as seen in the results below.

### 2.4.2 Results

Tables II, III and IV present the results obtained with the networks trained on Telugu consonant data.

Tables V, VI and VII present the results obtained for the networks trained on Telugu vowel modifier data.

Network	No.FM in the H <sub>1</sub> layer	No.FM in the H <sub>2</sub> layer	Data used for Training	Train Efficiency (%)	Data used for Testing	Test Efficiency (%)
1	5	10	23715	99.65	5156	90.16
2	5	12	23713	99.70	5156	89.97
3	5	16	23715	99.82	5156	90.16
4	6	10	23715	99.69	5156	90.01
5	6	12	23713	99.74	5156	90.05
6	6	16	23715	99.85	5156	90.14
7	7	10	23715	99.70	5156	90.01
8	7	12	23713	99.75	5156	89.99
9	7	16	23715	99.84	5156	90.08

**Table 2.11 Performance of Consonant Networks**

• FM : Feature Maps • H<sub>1</sub> : First Convolution layer • H<sub>2</sub>: Second Convolution layer

	1	2	3	4	5	6	7	8	9
1	1	0.954	0.993	0.993	0.959	0.992	0.992	0.956	0.992
2	0.954	1	0.954	0.965	0.996	0.957	0.965	0.995	0.958
3	0.993	0.954	1	0.991	0.960	0.992	0.990	<b>0.953</b>	0.991
4	0.993	0.965	0.991	1	0.969	0.993	0.999	0.964	0.993
5	0.959	0.996	0.960	0.969	1	0.961	0.970	0.997	0.962
6	0.992	0.957	0.992	0.993	0.961	1	0.992	0.957	0.997
7	0.992	0.965	0.990	0.999	0.970	0.992	1	0.965	0.992
8	0.956	0.995	0.953	0.964	0.997	0.957	0.965	1	0.959
9	0.992	0.958	0.991	0.993	0.962	0.997	0.992	0.959	1

**Table 2.11 Correlation among the Consonant Networks**

Network	Data used for Testing	Testing Efficiency (%)	
		Voting 1	Voting 2
Network <sub>3-8</sub>	5156	92.26	91.66

**Table 2.12 Ensemble Classifier Performance for Consonant Data**

Network	No.FM in the H <sub>1</sub> layer	No.FM in the H <sub>2</sub> layer	Data used for Training	Train Efficiency (%)	Data used for Testing	Test Efficiency (%)
1	5	10	23715	98.69	5156	89.16
2	5	12	23713	99.27	5156	89.41
3	5	16	23715	99.25	5156	89.85
4	6	10	23715	99.28	5156	89.29
5	6	12	23713	99.29	5156	89.23
6	6	16	23713	99.51	5156	89.22
7	7	10	23713	99.03	5156	89.02
8	7	12	23715	99.13	5156	89.78
9	7	16	23715	99.35	5156	89.29

Table 2.13 Performance of Vowel Modifier Networks

	1	2	3	4	5	6	7	8	9
1	1	0.964	0.995	0.998	0.957	0.956	0.963	0.995	0.991
2	0.964	1	0.965	0.968	0.997	0.995	0.997	0.955	0.961
3	0.995	0.965	1	0.993	0.958	0.948	0.964	0.997	0.981
4	0.998	0.968	0.993	1	0.960	0.952	0.967	0.994	0.977
5	0.957	0.997	0.958	0.960	1	0.995	0.995	0.945	0.961
6	0.946	0.995	0.948	0.952	0.995	1	0.996	<b>0.937</b>	0.953
7	0.963	0.997	0.964	0.967	0.995	0.996	1	0.955	0.961
8	0.995	0.955	0.997	0.994	0.945	0.937	0.955	1	0.977
9	0.991	0.961	0.981	0.977	0.961	0.953	0.961	0.977	1

Table 2.14 Correlation among Vowel Modifier Nnetworks

Network	Data used for Testing	Testing Efficiency (%)	
		Voting 1	Voting 2
Network <sub>6-8</sub>	5156	92.00	91.33

Table 2.15 Ensemble Classifier Performance for Vowel Modifier Data

## 2.5 Recognition Using Auto Encoders

### 2.5.1 Introduction and Motivation

It has been a long held belief in the field of neural network research that the composition of several levels of non-linearity would be the key to efficiently model complex relationships between variables and to achieve better generalization performance on

difficult recognition tasks. This viewpoint is motivated in part by knowledge of the layered architecture of regions of the human brain such as the visual cortex.

Yet, looking back at the history of multi-layer neural networks, their problematic non-convex optimization has for a long time prevented reaping the expected benefits of



going beyond one or two hidden layers. Consequently much of machine learning research has seen progress in shallow architectures allowing for convex optimization, while the difficult problem of learning in deep networks was left untouched.

Training a deep network to directly optimize only the supervised objective function by gradient descent, starting from random initialized parameters, does not work very well as we can only use this method to train shallow networks. The reason for that is that when we keep increasing the layers, back-propagation seems to become less efficient as the derivatives keep reducing and become very small by the time they reach the first layer. Hence, it is observed that the classification performance tends to reduce on adding layers and making the network deep.

### 2.5.2 How to Build Deep Networks?

It turns out that the problem is that the optimization function gets stuck in local optima. A logical approach to build Deep Neural Networks is to prevent the network from getting stuck into local minima; this is possible by choosing good values for the

starting weights. If the starting values of weights are already close to the region of

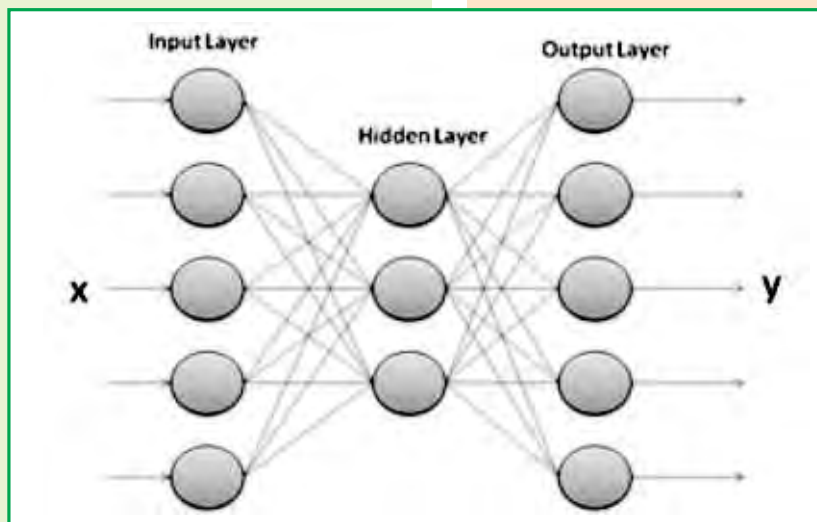


Fig 2.9. Sample Auto-encoder Neural Network

To extract interesting features from the Auto-encoder, we impose a sparsity constraint on the hidden layer, which makes the output vector of the hidden layer sparse, i.e. making sure that most of the nodes in the hidden layer are inactive most of the time. We define a sparsity parameter  $\rho$  ( $\rho \sim 0$ ), and restrict the average activation of every node in the hidden layer (averaged over all training examples) to be close to  $\rho$ . We do so by calculating  $\rho(i)$  corresponding to the activation of  $i$ th node in the hidden layer such that:

$$KL = \sum_{j \in 1:n} ac(i; j) = \rho(i)$$

where  $ac(i; j)$  corresponds to the activation of hidden node ' $i$ ' to training example ' $j$ ',  $n$  corresponds to the number of training examples and ' $m$ ' corresponds to the number of hidden layer nodes. We force  $\rho(i)$  to approach  $\rho$  by calculating the KL Divergence and minimizing it.

$$KL = \sum_{i \in 1:m} \rho * \log\left(\frac{\rho}{\rho(i)}\right) + (1-\rho) * \log\left(\frac{1-\rho}{1-\rho(i)}\right)$$

To minimize the KL divergence term, we add it to the net cost function (Ng, 2011)

$$Cost = \|d - y(x)\|^2 + \beta * KL + \lambda * \sum \|w\|^2$$

Where  $y(x)$  is the predicted output by the network,  $d$  is the desired output,  $\beta$  is the weight given to the sparsity term and  $\lambda$  is the regularization (weight decay) parameter.

### 2.5.3.2 Optimization:

We use limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimization algorithm to implement back-propagation in training autoencoders, using Mark Schmidt's minfunc package. min Func uses a quasi-Newton strategy, where limited-memory BFGS updates with Shanno-Phua scaling are

used in computing the step direction. In the line search, cubic interpolation is used to generate trial values, and the method switches to an Armijo back-tracking line search on iterations where the objective function enters a region where the parameters do not produce a real valued output.

### 2.5.3.3 Greedy Layer-wise stacking Auto-encoders

Greedy layer-wise pre-training overcomes the challenges of deep learning by introducing a prior in the weights to the supervised fine-tuning training procedure. The parameters are restricted to a relatively small volume of parameter space within a local basin of attraction generated by supervised fine-tuning cost function. Auto-encoders can be stacked to form a deep network by feeding the latent representation of layer below as input to the current layer. The unsupervised pre-training is done one layer at a time. Each layer is trained by minimizing the reconstruction of its input. Once the first  $k$  layers are trained, we can train the  $(k+1)^{th}$  layer because we can now compute the output of the  $k$ 'th layer and thus build an independent auto-encoder that reconstructs that output. Once all layers are pre-trained, the network goes through a second stage of training called fine-tuning. Here we consider supervised fine-tuning where we minimize prediction error on a supervised task. To this end, we first add a logistic regression layer that acts as a classification layer for the network. We then train the entire network as a simple feed-forward neural network using standard back-propagation algorithm.

### 2.5.3.4 Ensemble Classifier

We create an ensemble classifier by

combining various classifiers with least correlation in the performance on the validation set. We trained 10 different Deep Networks of different sizes on the training set. We ranked the networks on the basis of their performance on the validation set. We selected 4 networks among these 10 with maximum performance and least correlation amongst them. This was carried out by using a greedy approach wherein the performance of all the 10 networks on the validation set was calculated and we took all possible combinations of 4 networks and carried out majority voting to find the performance of the ensemble classifier over the validation set. The ensemble that gave the best performance over the validation set was chosen.

#### 2.5.4 Result

Following Tables present the result obtained after fine tuning the parameters using five-fold cross validation technique

Type	Train Size	Testing Size	Validation Size	Classes
C	21600	6000	1200	24
VM	13500	3750	750	15

#### Parameters used:

- Weight decay parameter ( $\lambda$ ) = 0.001,
- Weight of the sparsity penalty term ( $\beta$ ) = 4

-Sparsity parameter ( $\rho$ ) = 0.1

-Activation Functions:

- First two Hidden Layers use sigmoid activation functions
- The Output Layer uses soft-max function to perform classification

-The training was performed in batch mode and fine tuning back propagation with maximum number of iterations = 2000.

#### 2.5.4.1 Network performance with Consonant Dataset

The trained autoencoder can be visualized by plotting the extracted feature vectors as images. Fig.2.10 depicts the feature weights of the network with hidden layer size (24X24, 20X20). It represents the weights from the input layer to the first hidden layer, with each patch having a similar size as the input image and having total number of images same as the total number of hidden

nodes, where each image corresponds to a node in the first hidden layer.

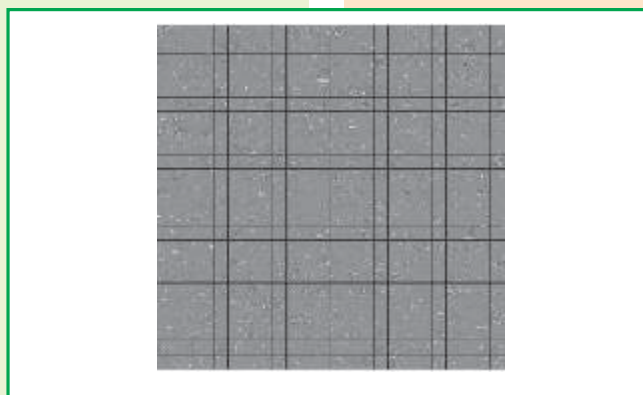


Fig 2.10. Feature weights of the network (24X24, 20X20).

Hidden Layer1	Hidden Layer2	Test Accuracy
24X24	20X20	94.25
20X20	14X14	91.5
14X14	7X7	89.5
30X30	28X28	94.25
20X20	16X16	92.3
14X14	10X10	90.8
18X18	10X10	91.25

Table 2.15 Performance using Two Hidden Layers.

Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Test Accuracy
24X24	20X20	16X16	94.8
24X24	14X14	10X10	91.75
20X20	16X16	10X10	92.25
20X20	10X10	7X7	90.5

Table 2.16 Performance using Three Hidden Layers.

#### 2.5.4.2 Performance with Ensemble Classifier.

We get a performance of 95.4% using the ensemble classifier by combining 4 networks (each two hidden layer) selected among 10 different networks. The selected 4 networks are with sizes {24X24, 20X20},

{20X20, 14X14}, {20X20, 16X16}, {14X14, 10X10}

#### 3.5.3.3 Network performance with Vowel Modifier Dataset

The trained auto-encoder can be visualized in a similar fashion as we did for the consonant dataset.

Hidden Layer1	Hidden Layer2	Test Accuracy
24X24	20X20	94.1
20X20	14X14	90.8
14X14	7X7	88.7
30X30	28X28	94.2
20X20	16X16	91.2
14X14	10X10	89.6
18X18	10X10	90.2

Table 2.17 Performance using Two Hidden Layers.

Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Test Accuracy
24X24	20X20	16X16	94.3
24X24	14X14	10X10	91.3
20X20	16X16	10X10	92.0
20X20	10X10	7X7	89.8

Table 2.18 Performance using Three Hidden Layers.

#### 2.5.4.4 Performance with Ensemble Classifier.

We get a performance of **94.8%** using the ensemble classifier that combines 4 networks (each two hidden layer) selected among 10 different networks. The selected 4 networks are with sizes {24X24, 20X20}, {20X20, 14X14}, {20X20, 16X16}, {14X14, 10X10}

### 3. Offline Recognition of Numerals, special Characters and Preclassification of Mixed Text Using PCA\_CNN-SVM

#### 3.1 Numeral Recognition Using PCA\_CNN-SVM

Data sets used in this study is MNIST data base of handwritten digits. MNIST data is a well known benchmark data set consisting of handwritten digit images. It has a training set of 60,000 examples, and a test set of 10,000 examples. MNIST images are size-normalized and centered in a fixed-size image. The number of output classes is 10 corresponding to the numeral 0 to 9 with an almost equal distribution for all the classes.

##### 3.1.1 Result

Network	Data used for Testing	Testing Efficiency (%)	
		<i>Voting 1</i>	<i>Voting 2</i>
Network <sub>3-8</sub>	10000	98.50	98.40

Table 3.1: Ensemble Classifier Performance on MNIST Data

#### 3.2 Special Character Recognition Using PCA\_CNN-SVM

Special Character's are a computer-representable character that is not alphabetic, numeric, or blank. Special Characters used in our study are shown in Table 3.2

1	\	back slash
2	/	forward slash
3	~	tilde
4	@	At
5	\$	Dollar
6	%	percent
7	^	Carat
8	&	ampersand
9	*	Asterisk
10	(	open parenthesis
11	)	close parenthesis
12	+	plus
13	<	less than
14	>	greater than
15	?	Question mark
16	[	open bracket
17	]	close bracket
18	{	open brace
19	}	close brace

Table 3.2: Special Characters



Special characters consists of either a single stroke or multiple strokes. Online stroke information of Special characters are normalized with leftmost and bottommost point at (0, 0) and main stroke height 1 and

### 3.2.1 Results

No.Data used for Train	Efficiency(%)	No.Data used for Test	Test(%)
800	100	580	95.86

### 3.3 Preclassification of Mixed Text Using PCA\_CNN-SVM

Preclassification seeks to classify the character in a Mixed\_Text (word, numerals, punctuations etc) into three categories:1) Telugu Character 2) Numeral 3) Special Character .

Class 1 – Telugu Character.

Class 2 – Numeral.

Class 3 – Special Character.

### 3.3.1 Results

Train Size	Efficiency (%)	Test Size	Efficiency (%)
15800 (10000(consonant)+5000(numeral) +800(special char)	100	8600 (5000 consonant + 3000 numerals + 600 special char)	99.29

Table 3.4: Preclassification Performance

## 4. Data Status

### 4.1 Annotation Data Summary

Annotation of handwriting is the process of labelling input data by the assignment of a text or numeric label to a selected set of words, characters and strokes for training for the purpose of a variety of handwriting analysis problems like Handwriting recognition. Manual annotation of large datasets is tedious,

then converted into binary images of size 56 x 56/28 x 28 pixels (2D images). These images are input to the PCA\_CNN-SVM. The index of the detected character is then used to display the character.

e.g. Address field

For Pre-classification we have used a combination of Telugu characters,digits ( MNIST database) and special characters. Train set consist of 10,000 Telugu CV type characters,5000 numerals and 800 special characters. Test set consist of 5000 Telugu CV type characters ,3000 numerals and 600 special characters.

expensive, and error prone process, especially at character and stroke level and tools for automating the process of annotation are useful for large corpora.

The creation of annotated datasets of handwriting is generally comprised of sequential data collection and annotation phases. In the data collection phase, writers with specific skills are recruited for

In word level data collection, each writer was asked to write a list of words in different fields of an electronic form and the ink from each field was automatically grouped into “words” and word-level annotation was done. The annotated word data was manually checked by array screening for accuracy. The character label is identified by matching the unlabeled character with available character templates and available information related to order of occurrence of characters for each word.

We have 17565 characters collected as isolated characters and annotated at both character and stroke level. We have class A word data with about 18,000 words and 55,500 characters annotated at word and character level. The class A word data has been converted into character data and annotation of this character data at stroke level is underway using the principle of matching unlabeled stroke data to available stroke templates and order of occurrence of strokes for each character.



## 4.2 Data Corpus

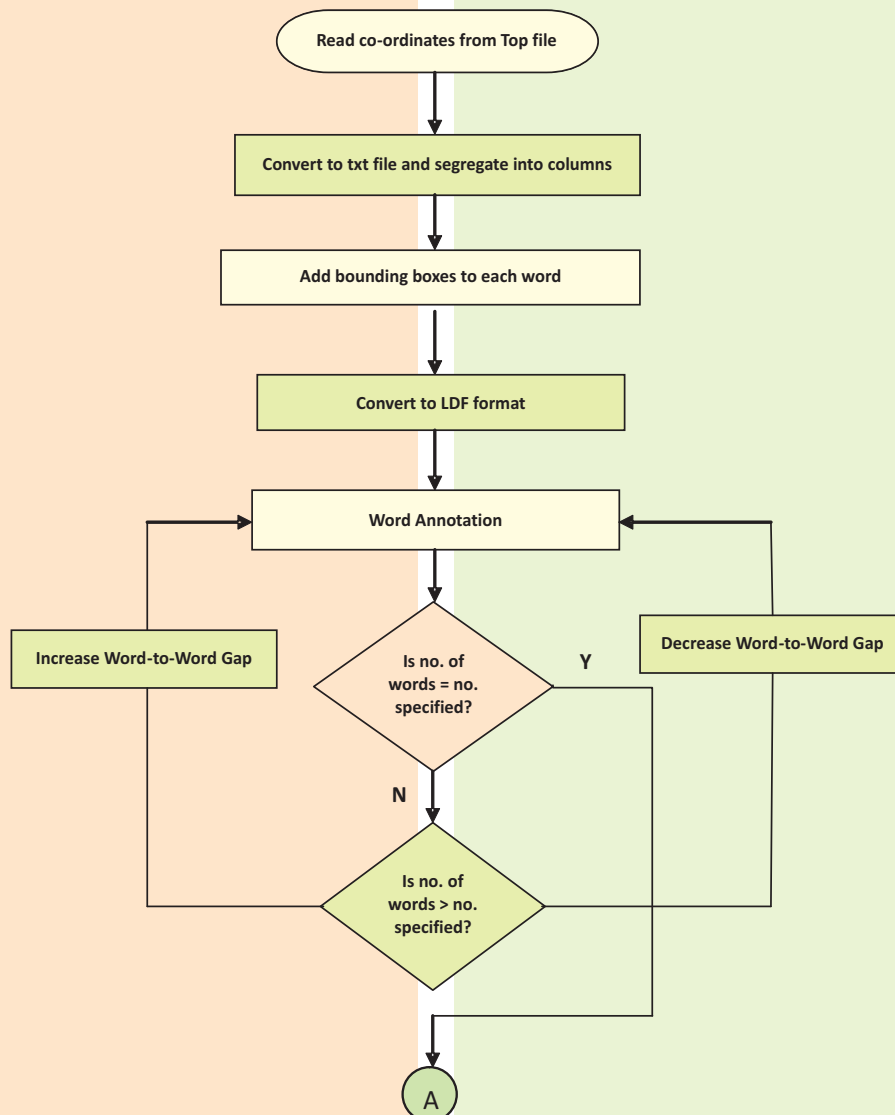
We have 2 sets of data - one collected as characters and other collected as a group of

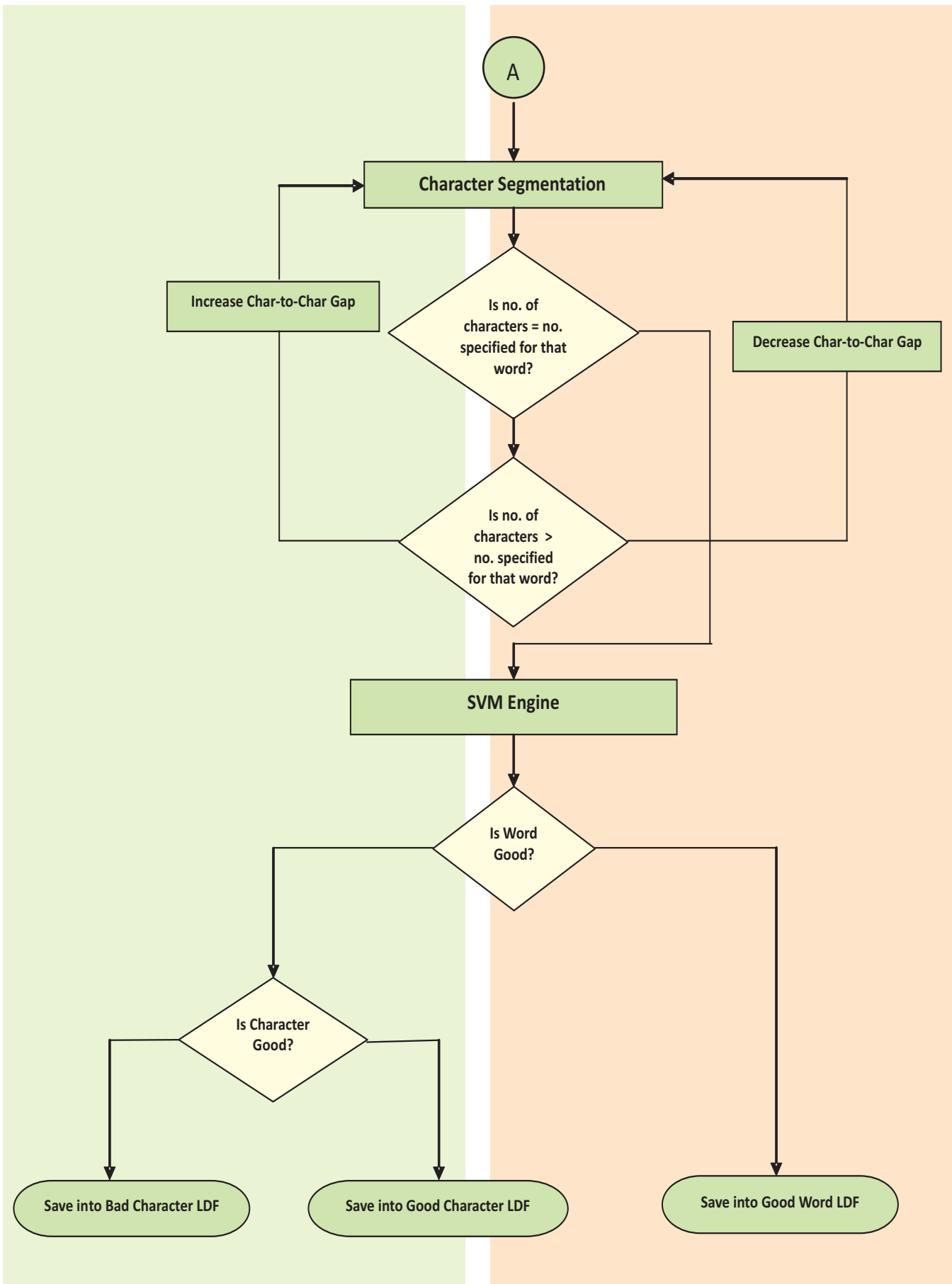
words of 16 pages. The template for word data collection is included in the appendix.

Word Level		
No. of Writers	715	IITM + IIITH
TOP Files	5731	16 Templates
CLASS A Word Data	31968	Annotated at Word and Character Level
CLASS A Character Data	10 2721	Annotated at Character Level. Stroke Level annotation under progress.

Table 4.1: Data Corpus

## 4.3 Steps for Data Labeling and Collection:





#### 4.3.1 Read co-ordinates from TOP file

The point co-ordinates from the Note-Taker are saved as. TOP file as pressure, validity, horizontal and vertical high and low points. These are transformed into co-ordinate points and pen on/off points. The strokes are determined by the pen off points.

#### 4.3.2 Conversion to txt files and segregate into columns

The TOP file page is segmented into columns based on the associated template page. The header and footer information that does not correspond to the word list is eliminated.

#### 4.3.3 Add bounding boxes to each word

Each column is segmented into rows based on the vertical gap between words.

#### 4.3.4 Conversion to LDF format

The words are saved into the prescribed LDF structure format.

#### 4.3.5 Word Annotation

The word to word gap is altered based on the number of words per page specified in the template to obtain the same and labelled at the word level.

#### 4.3.6 Character Segmentation

The words are labelled at the character level by splitting the words into characters based on the character to character gap till the specified numbers of characters are obtained for that word. They are then annotated at the character level.

#### 4.3.7 SVM Engine Truthing

The word structure annotated at the character level is sent to the support vector

machine for character recognition. If all the characters of a word are correctly recognized, then it is considered that the word is recognized.

#### 4.3.8 Saving in LDF

The characters and words that are recognised are saved as separate LDF (Layered Data Format) files. The unrecognised words and characters are also saved into LDF files for future data analysis and extraction.

### 5 Synthesis of characters

It is clear from the charts above that there is unequal distribution of characters across different classes and few of the CV combinations of characters are absent. One approach to obtaining data to include missing CV combinations and obtaining equal distribution amongst all classes is data collection using writers. Another approach is synthesis of natural looking characters.

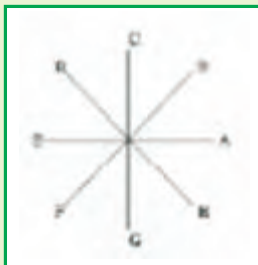
The character and word sets used for data collection were designed such that it included all the possible strokes in Telugu script. Using this complete stroke list, any V, CV, CCV type character can be synthesized, the procedure of which is explained below.

#### 5.1 Spatio-structural Features

Spatio-structural features capture information in the structural features present in a stroke and their spatial location within the stroke. Spatio-structural features capture information of the spatial proximity as opposed to temporal proximity of data points. Hence they are not sensitive to writing direction or style as spatiotemporal or spectral features.



Feature	Description	Shape
I	0 bump, negative Curvature	
J	-45° bump, negative Curvature	
K	0 bump, negative Curvature	
L	+45° bump, negative Curvature	
M	0 bump, positive Curvature	
N	-45° bump, positive Curvature	
O	0 bump, positive Curvature	
P	+45° bump, positive Curvature	
Q	Cusp	
R	Dot	



**Fig 5.1 The 18 shape based features for representing strokes. These include the dot point (R), line terminals (A-H), bump points (I-P) and cusp point (Q)**

The global shape of a character can be determined by a set of local shapes as described as above. The local shapes which are few in number can be combined variously to give rise to a great diversity of characters.

## 5.2 ALGORITHM

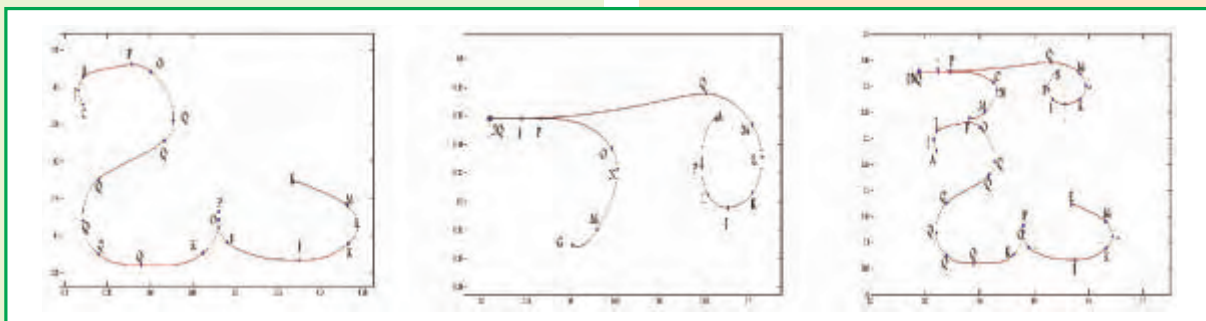
Shape feature points are extracted for each stroke. The combination of strokes for each character is known a priori. For each character a rule is made indicating the order and the procedure in which the strokes, one main stroke and one or more auxiliary strokes, come together to form a character.

There are different scenarios possible.

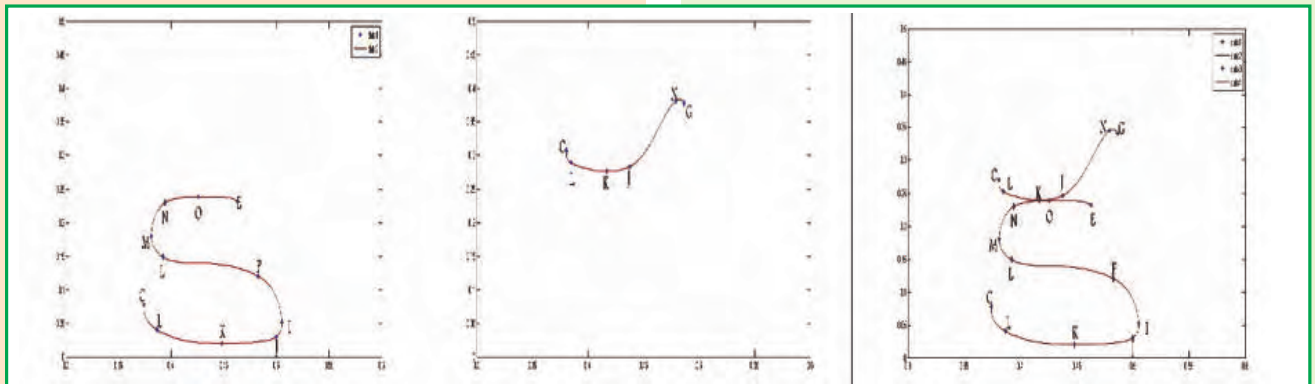
The auxiliary stroke attaches to the main stroke and the shape points on both overlap at the point of attachment. The translation required to move the shape feature point on the auxiliary stroke to overlap with the shape point on main stroke is calculated and the auxiliary stroke is translated by the same amount to form the character.

The auxiliary stroke attaches to the main stroke and there is no overlap of shape feature points at the point of attachment. . The point of attachment is measured as a fraction of distance between two shape points calculated from the first shape point. The x coordinate and y coordinate values corresponding to the point of attachment are determined and the translation required is determined. The auxiliary stroke is translated to form the character.

The auxiliary stroke does not attach to the



**Fig 5.2 The vowel 'au' can be synthesised from strokes 'o' in first figure and 'au\_ym' in second figure. 'au\_ym' is translated such that shape point 'G' on 'au\_ym' overlaps with shape point 'P' on 'o'.**



**Fig 5.3** The vowel 'ka' can be synthesised from strokes 'ka\_base' in first figure and 'tick' in second figure. 'tick' is translated such that shape point 'K' on 'tick' is attached between shape points 'O' and 'N' on 'ka'.

main stroke anywhere. The main stroke is normalised to a height of one. The auxiliary stroke is placed above or below the main stroke between two shape points. The distance between the two shape points above/below which the auxiliary stroke is placed is determined. The height above or below this point where the auxiliary stroke will be placed is measured as a fraction of the main stroke height. The translation required to move the auxiliary stroke is calculated.

### 5.3 Distortion of Online Data:

In order to improve the performance of the recognition engine, a large training dataset is required. By considering transformation-invariance property, distortion can be used to expand the dataset.

Distortion of Online Character or Stroke data is performed by adding noise to the low frequency component of the x and y points of a character or stroke data.

The steps involved in distortion of the stroke data is as follows:

1. Take Fast Fourier Transform (FFT) of the x and y points (stroke data).
2. Generate a Random Noise of range (-1, 1).

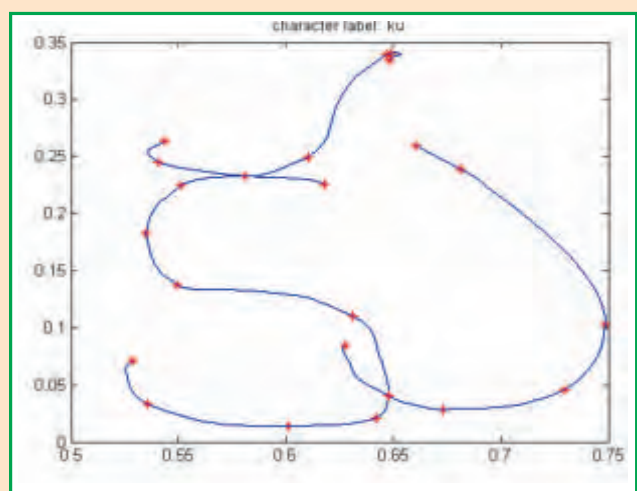
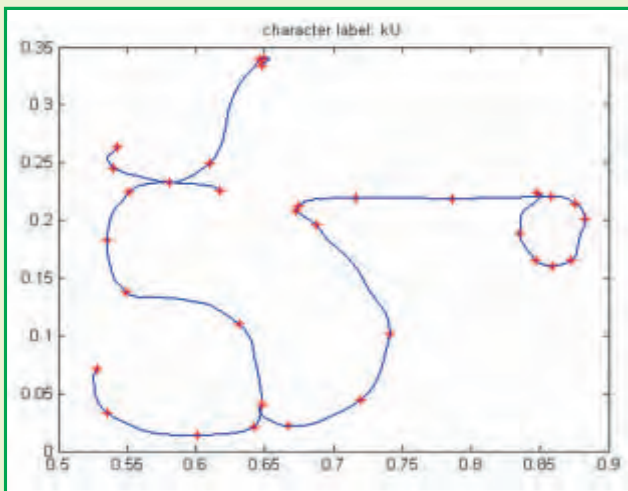
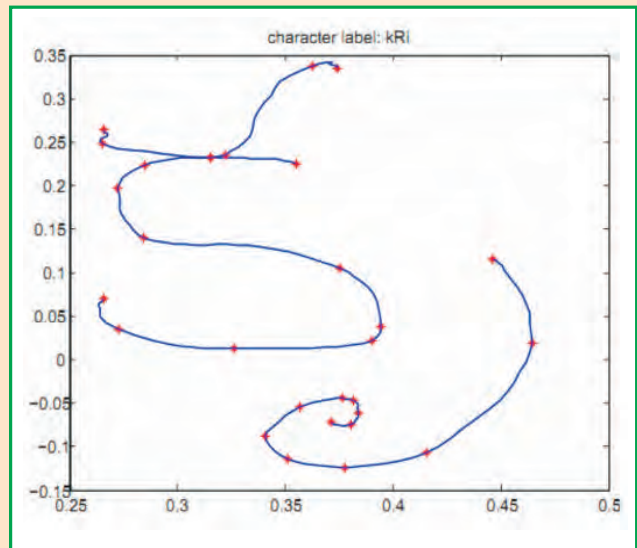
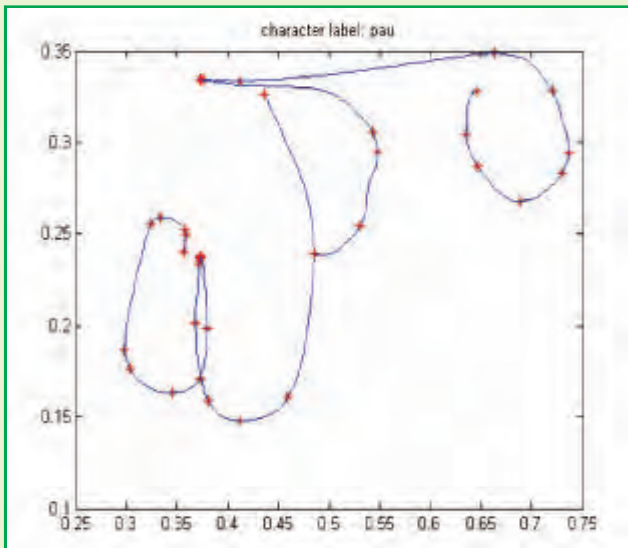
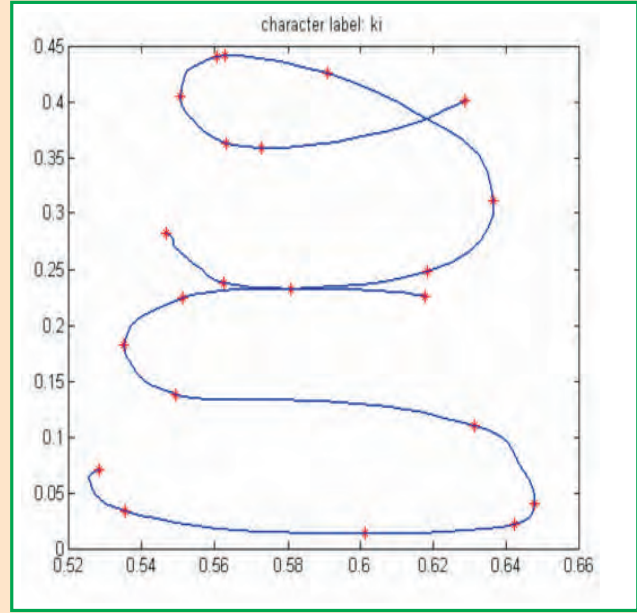
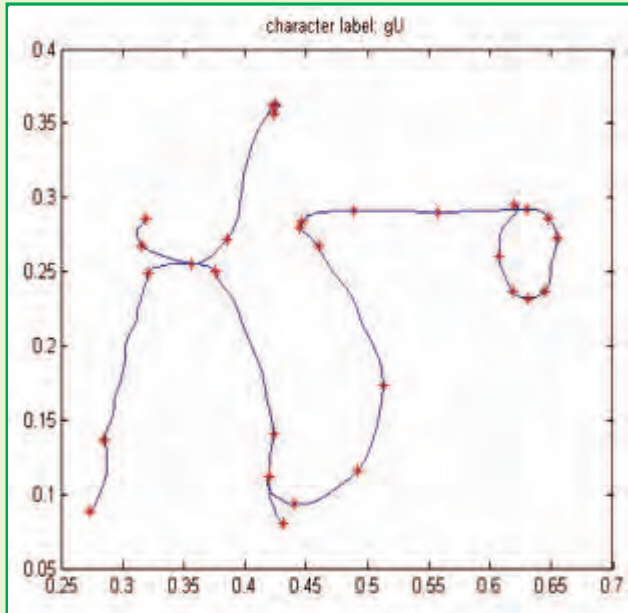
3. A scaling window is required to add noise to the frequency component.
4. The scaling window should be developed in such a way, that the high percentage of noise should be added to the low frequency components and low percentage or no noise should be added to the high frequency components.
5. Multiply the scaling window with the random noise and with certain percentage ( $p=50\%$  for x and  $q=10\%$  for y) of Fourier Transform Coefficients. This resultant product is added with FFT components.
6. Take the inverse Fast Fourier Transform for the result obtained in the previous step.
7. The real value of the result will be a distorted version of the original data.

$$\begin{aligned} X's \text{ FFT coefficient} &+ (p * \text{noise}) \\ Y's \text{ FFT coefficient} &+ (q * \text{noise}) \end{aligned}$$

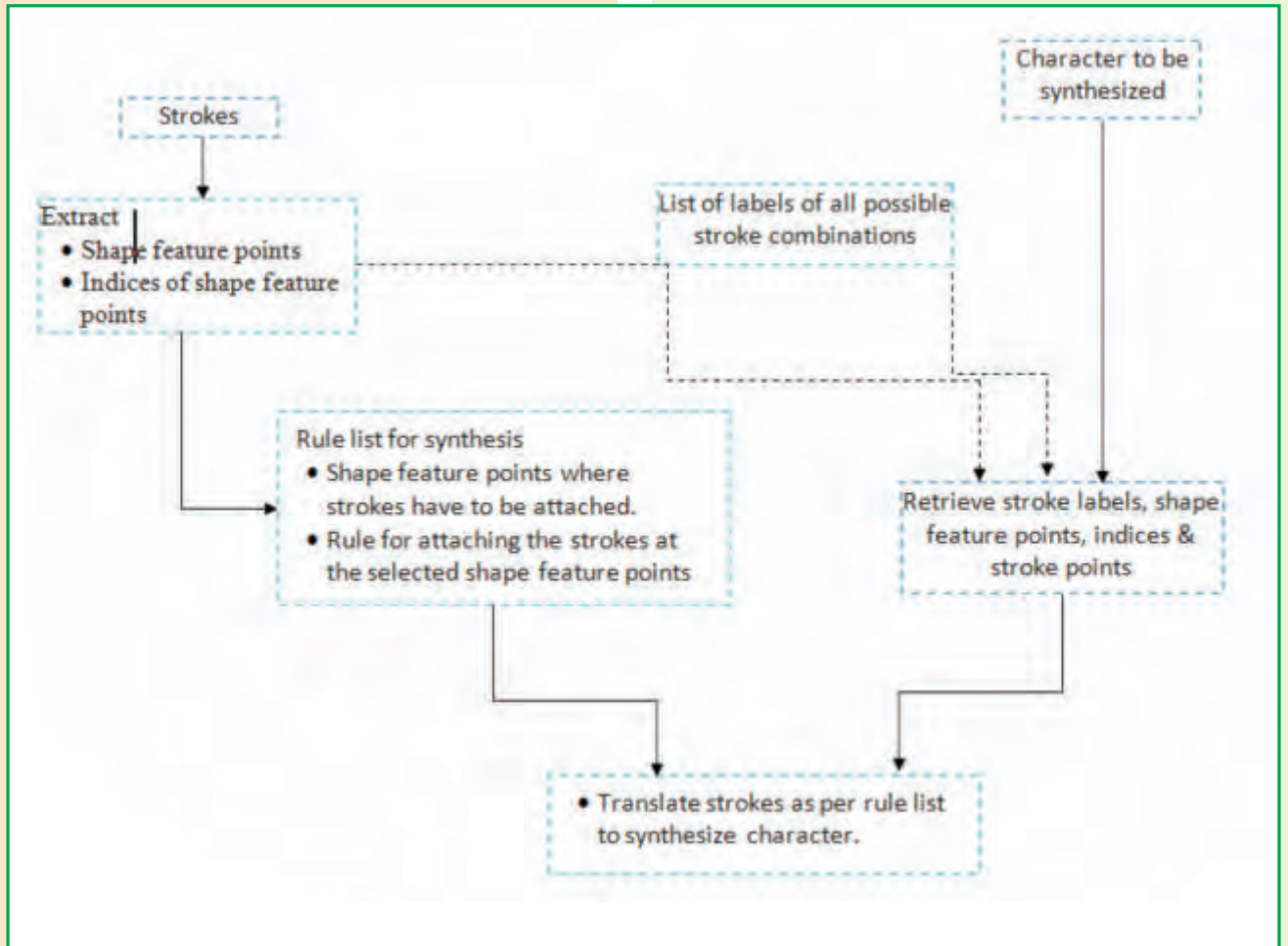
### 5.4 Synthesis of Distorted Characters:

The distorted characters have been synthesised using same set of algorithms. The distorted data is generated using frequency filtration concept. Frequency filtration is described above.

### Examples of Distorted Characters:-



### 5.5 Flow Diagram for Synthesis Algorithm:



#### Characters Synthesised:

Synthesized Characters	Total data
Without distortion	392
With distortion	392

#### Applications:

This data could be useful for augmenting the dataset.

\*\*\*