Chapter 5. Classification of Symbols of Gujarati script using General Regression Neural Network

5.1 Introduction

Artificial neural networks, due to their learning capabilities, are often used in the field of character recognition. There are several previous works that use various ANN architectures for glyph recognition in printed Indian scripts [3, 4, 7, 8]. All these approaches employ architectures using iterative learning techniques like Back propagation [3, 4, 9, 10], RBF [9], Dynamic Neural Networks (DNN)[8]. The basic disadvantage of these techniques is that these algorithms take a large number of iterations to converge to desired solution. General Regression Neural Network, in comparison is a single pass neural network architecture, and converges much faster to the desired solution. General Regression Neural Networks (GRNN) are first proposed for speech recognition [21] and used there with good success. However it is for the first time here that it is being used as a classifier for the Gujarati Character Recognition.

In this chapter, we discuss a neural network architecture based on the Statistical learning theory discussed in chapter-2. This neural network is usually much faster to train than the traditional multilayer perceptron network. The organization of this chapter is as follows:

This chapter is divided into six sections. After this introduction, the second section introduces two neural network architectures which are based on Statistical learning theory. The third section describes the General Regression Neural Network architecture from the Mathematical point of view while the computational aspects of the architecture are discussed in the fourth section. In the fifth section, we present the experimental details of the use of GRNN for classification of Gujarati symbols in the development of an OCR system for the Gujarati script [30]. We summarize the chapter in the sixth section.

5.2 Introduction to Probabilistic and General Regression Neural Networks

The use of Statistical learning process in the field of neural networks has been proposed by Donald F. Specht in 1990 [38]. He proposed a method to formulate the weighted-neighbor method in the form of a neural network. He called this a "Probabilistic Neural Network (PNN)". In 1991, he proposed a neural network architecture called "General Regression Neural Network (GRNN)" [21]. PNN and GRNN have similar architectures, but there is a fundamental difference: Probabilistic networks perform classification where the target variable is categorical, whereas general regression neural networks perform regression where the target variable is continuous.

PNN and GRNN networks have the following advantages and disadvantages compared to Multilayer Peceptron:

- It is usually much faster to train a PNN/GRNN network than a multilayer perceptron network.
- PNN/GRNN networks often are more accurate than multilayer perceptron networks.
- PNN/GRNN networks are relatively insensitive to outliers (wild points).
- PNN networks generate accurate predicted target probability scores.
- PNN networks approach Bayes optimal classification.
- PNN/GRNN networks are slower than multilayer perceptron networks at classifying new cases.
- PNN/GRNN networks require more memory space to store the model.

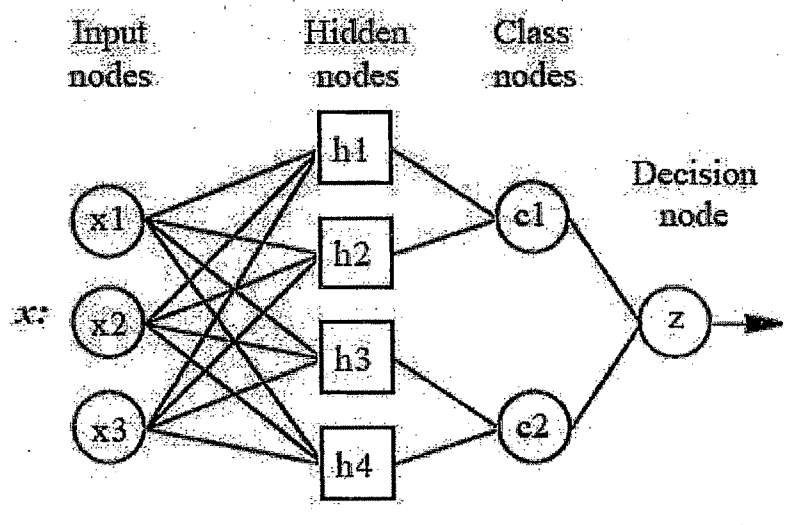
Although the implementation is very different, probabilistic neural networks are conceptually similar to K-Nearest Neighbor (K-NN) models. The basic idea is that a predicted target value of an item is likely to be about the same as other items that have close values of the predictor variables.

A probabilistic neural network builds on this foundation and generalizes it to consider all of the other points. The distance is computed from the point being evaluated to each of the other points, and a radial basis function (RBF) (also called a kernel function) is applied to the distance to compute the weight (influence) for each point. The radial basis function is so named because the radius distance is the argument to the function.

5.3. Architecture of a PNN/GRNN Network

The probabilistic neural network uses a supervised training set to develop distribution functions within a pattern layer. These functions, in the recall mode, are used to estimate the likelihood of an input feature vector being part of a learned category, or class. The learned patterns can also be combined, or weighted, with the a priori probability, also called the relative frequency, of each category to determine the most likely class for a given input vector. If the relative frequency of the categories is unknown, then all categories can be assumed to be equally likely and the determination of category is solely based on the closeness of the input feature vector to the distribution function of a class.

The basic architecture of both the neural networks with one neuron in the output layer can be depicted as below:



[Fig. 5.1 Architecture of PNN / GRNN]

All PNN/GRNN networks have four layers:

1. Input layer — There is one neuron in the input layer for each training vector. The input neurons (or processing before the input layer) standardize the range of input values by subtracting the median and dividing by the interquartile range. The input neurons then feed the values to each of the neurons in the hidden layer.

2. Hidden layer — This layer has one neuron for pair in the training data set. The neuron stores the values of the predictor variables for the case along with the target value. When presented with the **X** vector of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron's center point and then applies the RBF kernel function using the spread value (s). The resulting value is passed to the neurons in the pattern layer.

3. Pattern layer / Summation layer — The next layer in the network is different for PNN networks and for GRNN networks. For PNN networks there is one pattern neuron for each category of the target variable. The actual target category of each training case is stored with each hidden neuron; the weighted value coming out of a hidden neuron is fed only to the pattern neuron that corresponds to the hidden neuron's category. The pattern neurons add the values for the class they represent (hence, it is a weighted vote for that category). For GRNN networks, there are only two neurons in the pattern layer. One neuron is the denominator summation unit and the other is the numerator summation unit. The denominator summation unit adds up the weight values coming from each of the hidden neurons. The numerator summation unit adds up the weight values multiplied by the actual target value for each hidden neuron.

4. Decision layer — The decision layer is different for PNN and GRNN networks. For PNN networks, the decision layer compares the weighted votes for each target category accumulated in the pattern layer and uses the largest vote to predict the target category. For the GRNN networks, the decision layer divides the value accumulated in the numerator summation unit by the value in the denominator summation unit and uses the result as the predicted target value.

The pattern layer represents a neural implementation of a version of a Bayes classifier [chaptet-2 section 2.2.1], where the class dependent probability density functions are estimated using a Parzen estimator (discussed in the section 5.4). This approach provides an optimum pattern classifier in terms of minimizing the expected risk (training error) of wrongly classifying an object. With the estimator, the approach gets closer to the true underlying class density functions as the number of training samples increases, so long as the training set is an adequate representation of the class distinctions.

In the pattern layer, there is a processing element for each input vector in the training set. Normally, there are equal amounts of processing elements for each output class. Otherwise, one or more classes may be skewed incorrectly and the network will generate poor results. Each processing element in the pattern layer is trained once. An element is trained to generate a high output value when an input vector matches the training vector. The training function may include a global smoothing factor to better generalize classification results. In any case, the training vectors do not have to be in any special order in the training set, since the category of a particular vector is specified by the desired output of the input. The learning function simply selects the first untrained processing element in the correct output class and modifies its weights to match the training vector.

The pattern layer operates competitively, where only the highest match to an input vector wins and generates an output. In this way, only one classification category is generated for any given input vector. If the input does not relate well to any patterns programmed into the pattern layer, no output is generated.

The Parzen estimation can be added to the pattern layer to fine tune the classification of objects, This is done by adding the frequency of occurrence for each training pattern built into a processing element. Basically, the probability distribution of occurrence for each example in a class is multiplied into its respective training node. In this way, a more accurate expectation of an object is added to the features which make it recognizable as a class member.

Training of the probabilistic neural network is much simpler than training a Multilayer Perceptron with back-propagation. However, the pattern layer can be quite huge if the distinction between categories is varied and at the same time quite similar *is* special areas. There are many proponents for this type of network, since the groundwork for optimization is founded in well known, classical mathematics.

5.4 Mathematical aspects of GRNN

Let N is the number of realizations of the set T and m_0 is the number of samples in each realization. Consider a training set $T = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$ which are realizations of a random vector **X**, as discussed in the section 2.2.3 of the chapter 2.

Consider a training set T. Ordinarily we don't have knowledge of the exact functional relationship between χ and Y [18]. We may proceed as follows:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad i = 1,2,\ldots N \tag{5.1}$$

Equation (5.1) , is known as a regressive model, where $f(.)$ is a deterministic function of its argument vector and ε is a random expectational error, that represents our "ignorance" about the dependence of Y on χ. Properties of expectational error is shown below:

Properties of Expectational error ε:

(i) The mean value of the expectational error ε, given any input **x**, is zero i.e.
$E[\varepsilon \mid \mathbf{x}] = 0$

(ii) The regression $f(\mathbf{x})$ is equal to the conditional mean of y given **x** (i.e. the regression of y on **x**) as shown by $f(\mathbf{x}) = E(y \mid \mathbf{x})$

From property (ii), using the formula for the expectation of a random variable, we may write

$$f(\mathbf{x}) = \int_{-\infty}^{\infty} y f_Y(y \mid \mathbf{x}) dy$$

Where $f_Y(y \mid \mathbf{x})$ is the conditional probability density function (*pdf*) of Y, given **x**.

From probability theory, we have

$$f_Y(y/\mathbf{x}) = \frac{f_{\mathbf{X},Y}(\mathbf{x},y)}{f_{\mathbf{X}}(\mathbf{x})}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is a probability density functions of **x** and $f_{X,Y}(\mathbf{x},y)$ is a joint probability density functions of **x** and y.

$$\text{Therefore } f(\mathbf{x}) = \frac{\int_{-\infty}^{\infty} y f_{\mathbf{X},Y}(\mathbf{x},y)dy}{f_{\mathbf{X}}(\mathbf{x})}$$

Our particular interest is in a situation where the joint probability density function $f_{\mathbf{X},Y}(\mathbf{x},y)$ is unknown. All that we have available is the training set, $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. To estimate $f_{\mathbf{X},Y}(\mathbf{x},y)$ and therefore $f_{\mathbf{X}}(\mathbf{x})$, we may use a nonparametric estimator known as the Parzen – Rosenblatt density estimator. Basic to the formulation of this estimator is a kernel, denoted by K(**x**), which has properties similar to those associated with a probability density function:

- The kernel K(**x**) is a continuous, bounded and real valued function of **x** and is symmetric about the origin, where it attains its maximum value
- The total volume under the surface of the kernel K(**x**) is unity; that is, for an *m*-dimensional vector **x**,

$$\int_{R^m} K(\mathbf{x})dx = 1 \tag{5.2}$$

Assuming that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are independent vectors and identically distributed (each of the random variables has the same probability distribution as the others), we may formally define the Parzen – Rosenblatt density estimate of $f_{\mathbf{X}}(\mathbf{x})$ as :

$$\hat{f}_{\mathbf{X}}(\mathbf{x}) = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \quad \text{for } \mathbf{x} \in R^{m_0} \tag{5.3}$$

where the smoothing parameter h is a positive number called bandwidth or simply width; h controls the size of the kernel.

An important property of the Parzen – Rosenblatt density estimator is that it is a consistent estimator (i.e. asymptotically unbiased) in the sense that if $h = h(N)$ is chosen as a funciton of N such that

$$\lim_{N\to\infty} h(N) = 0$$

then, $$\lim_{N\to\infty} E[\hat{f}_{\mathbf{X}}(\mathbf{x})] = f_{\mathbf{X}}(\mathbf{x})$$

In a similar manner to that described in equation (5.3), we may write the Parzen-Rosenblatt density estimate of the joint probability density function $f_{\mathbf{X},Y}(\mathbf{x}, y)$ as

$$\hat{f}_{\mathbf{X},Y}(\mathbf{x}, y) = \frac{1}{Nh^{m_0+1}} \sum_{i=1}^{N} K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) K(\frac{y-y_i}{h}) \qquad \text{for } \mathbf{x} \in R^{m_0},\ y \in R \qquad (5.4)$$

Multiplying equation (5.3) with y and integrating with respect to y, we get $f_{\mathbf{X}}(\mathbf{X})$ of equation (5.4)

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0+1}} \sum_{i=1}^{N} K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) \int_{-\infty}^{\infty} yK(\frac{y-y_i}{h})dy$$

Let $z = (y - y_i)/h \quad \Rightarrow y = y_i + hz$

$dz = dy/h \quad \Rightarrow dy = hdz$

Substituting the value of z in the right hand side of the equation, we obtain

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0+1}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \int_{-\infty}^{\infty} (y_i + hz)K(z)hdz$$

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \int_{-\infty}^{\infty} (y_i + hz)K(z)dz$$

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \left[\int_{-\infty}^{\infty} y_i K(z)dz + h\int_{-\infty}^{\infty} zK(z)dz\right]$$

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \left[y_i \int_{-\infty}^{\infty} K(z)dz + h\int_{-\infty}^{\infty} zK(z)dz\right]$$

but from equation (5.2) the first integral on the right becomes 1, therefore we get,

$$\int_{-\infty}^{\infty} y\, \hat{f}_{\mathbf{X},Y}(\mathbf{x}, y)dy = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \left[y_i + h\int_{-\infty}^{\infty} zK(z)dz\right]$$

From the property (i) of the kernel (symmetric about origin), K(Z) is an even function and Z is an odd function. Therefore the second integral becomes zero. So we get,

$$\int_{-\infty}^{\infty} y\,\hat{f}_{\mathbf{X},Y}(\mathbf{x},y)dy = \frac{1}{Nh^{m_0}}\sum_{i=1}^{N} y_i K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \tag{5.5}$$

From (5.3) and (5.5), we obtain estimate of the regression function $f(\mathbf{X})$ after canceling common terms as :

$$\mathrm{F}(\mathbf{x}) = \hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} y_i K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}$$

If we take Gaussian kernel i.e. $K(\mathbf{x}) = e^{-\mathbf{x}^2}$, we obtain,

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} y_i \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{N} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)} \tag{5.6}$$

where $D_i^2 = (\mathbf{x}-\mathbf{x}_i)^T(\mathbf{x}-\mathbf{x}_i)$ and σ is the standard deviation.

$\hat{f}(\mathbf{x})$ can be visualized as a weighted average of all observed values y_i, where each observed value is weighted exponentially according to its Euclidean distance from **x**.

The theory of General Regression Neural Networks discussed above is pertaining to only neuron in the output layer. The same technique can be applied for the multiple neurons in the output layer also. Therefore the technique can be generalized as shown below:

Let w_{ij} be the target output corresponding to input training vector $\mathbf{x}_i$ and j^{th} output node out of the total p. Again let $\mathbf{C}_i$ be the centers chosen from the random vector **X**. Then

$$y_i = \frac{\sum_{i=1}^{n} w_{ij} h_i}{\sum_{i=1}^{n} h_i} \tag{5.7}$$

where n be the number of patterns in the training set.

The estimate y_j can be visualized as a weighted average of all the observed values, where each observed value is weighted exponentially according to its Euclidean distance from input vector **x** and n is the number of patterns available in the input space.

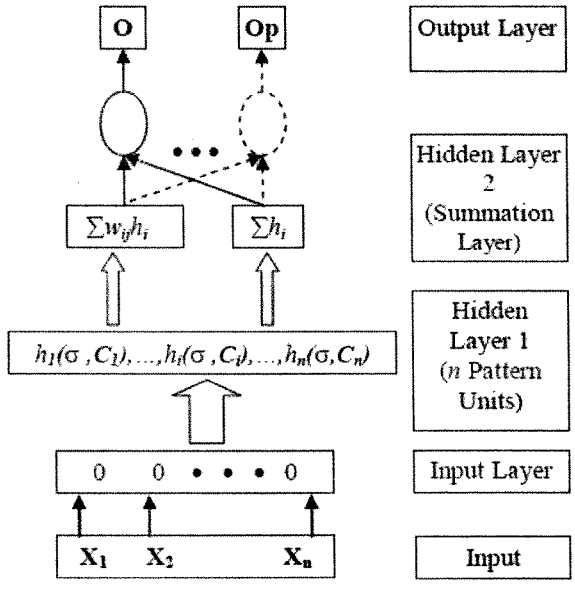
$$\text{with } h_i = h_i(\sigma, \mathbf{C}_i) = \exp\left(-\frac{D_i^2}{2\sigma^2}\right) \tag{5.8}$$

$$\text{Where, } D_i^2 = (\mathbf{x} - \mathbf{C}_i)^T (\mathbf{x} - \mathbf{C}_i)$$

5.5 Computational aspects of GRNN

According to (5.7) and (5.8) the topology of a GRNN (figure-5.2) consists of:

- The input layer (input cells), which is fully connected to the pattern layer
- The pattern layer which has one neuron for each pattern. It computes the pattern functions $h_i(\sigma, \mathbf{C}_i)$ which is expressed in (5.8) using the centers $\mathbf{C}_i$ and spread σ.
- The summation layer which has two units N and D. The first unit has input weights equal to $\mathbf{h}_i$, then it computes the numerator N by summation of the exponential terms multiplied by w_{ij} (j^{th} output neuron) associated with $\mathbf{x}_i$. The second unit has input weight equal to 1, then the denominator D is the summation of exponential terms alone.
- Finally the output unit divides N by D to provide the prediction result.



[Fig. 5.1 GRNN architecture]

In nutshell, the algorithm can be described as below:

When a new input pattern(no of elements = p) is exposed to the network as shown in figure 5.1, hidden layer 1 computes n (number of patterns) values of h using equation (5.8). Using those values of h, Hidden layer 2 computes p pairs of numerator N and denominator D. Each pair is fed to corresponding output neuron and output is computed by dividing N by D.

The choice of the smoothing parameter σ is very important. When σ is small, only a few samples plays a role in the estimation of output. If it is large then even distinct neighbors affect the estimation of output. In the extreme case, when σ goes to infinity $\hat{f}(\mathbf{x})$ in equation (5.6) is the average of all y_i, independent of the input.

5.5. Classification using General Regression Neural Networks

Artificial neural networks, due to their learning and generalization capabilities, are often used as classifiers in the field of character recognition. There have been several attempts of using different neural network architectures for Indian scripts [1,3,7,8,9,10,16,17]. However, there has been no earlier work on the use of General Regression Neural Networks (GRNN) [21] for the classification purpose in OCR of printed Indian scripts. This Neural Network architecture does not require any kind of training; it just estimates the expected output of a test pattern by considering input patterns. Therefore this architecture of GRNN is sometimes referred to as a one-step run architecture. Thus due to its generalization capability and no requirement for training, it produces classification results very fast. We are motivated to try this approach for character recognition since researchers have attempted speech recognition using GRNN [20] and reported high recognition accuracy.

We have applied the GRNN architecture for the classification of the glyphs of upper, middle and the lower zones in the images of printed Gujarati characters. Three separate networks have been constructed for the symbols of the three zones. Thus, this architecture has been employed to perform the most comprehensive set of experiments among all our attempts, for the Gujarati character recognition.

As discussed in the chapter 2, the characters of Gujarati script can be visualized as occupying three vertical zones viz. lower, middle and upper. The upper and lower zone glyphs are mostly made up of vowel modifiers while all other character components are contained in the middle zone. We have seen the recognition accuracies obtained for the lower and middle zone symbols using the multilayer perceptron architecture of neural networks in chapter 4.

The data base for recognition of glyphs is made up of 4173 images of the 119 types of middle zone symbols, 210 images of 4 types of lower zone modifiers and 269 images of 15 types of upper zone modifiers. The images are first normalized to a standard size of 32 x 32 pixels. Then the Daubechie's D4 discrete wavelet transform is applied on each image. The 16 x 16 low-low coefficients that result from this transform are then extracted as the features for the images capturing their important characteristics. These features are then exposed to the three GRNN networks for training and testing the networks. The details of these computations are provided below:

In the experiments presented here, we have constructed three networks: the first network is for middle zone symbols, the second network is for the lower zone symbols and third one is for upper zone symbols of Gujarati script (refer section 2.4). Each of these networks has 256 input neurons and as many output neurons as the number of classes (types) of the respective zone. An implentation of the GRNN architecture that can be used to realize these three networks has been developed as a java class. The class has attributes for the patterns, values of the gaussian function, center indices etc and methods for computing the gaussian matrix, numerator and denominator values for each category, generate the network output etc.

Table 1 gives the details about the data collection. In order to ensure comparability, we have tried to keep the set of samples for consonants and numerals almost same as the one referred in [4]. Total 4173 samples, covering 119 classes of middle zone glyphs out to about 250 total middle zone glyphs, are collected.

Table 1. Dataset Details

Sr. No.	Category	No. of Fonts	Font Sizes	Style
1	Consonants (37 symbols)	4	11 to 15	Regular, Bold, Italic
2	Independent Vowels (5 symbols)	3	11 to 15	"
3	Conjuncts (67 symbols)	2	11 to 15	Regular, Bold
4	Numerals (10 symbols)	3	11 to 15	Regular, Bold, Italic

Dependent vowel modifiers (matras) were deliberately not considered in this study as they need to be handled separately. Subset of 2802 symbols was used as training set or set of known information and remaining 1371 symbols were used as testing set. The network has achieved 97.59% of recognition accuracy with the learning rate 0.5.

The second network is used to classify the lower zone symbols [chapter 4] of the Gujarati script. We have taken 4 widely used lower zone symbols in this experiment and collected 200 samples. Out of the collection of 210 samples, 74 patterns are accommodated in the training set the remaining 136 patterns are taken in the testing set. Thus in this network, we have used 256 input neurons and 4 output neurons and achieved 97% of recognition accuracy with the learning rate 0.5.

The third network classifies the symbols of upper modifiers of the Gujarati script. We have collected 15 widely used symbols in our experiment. The dataset of upper modifiers consists of 269 patterns. Out of these, 154 selected randomly for the training set and the remaining 115 considered in the testing set. This network is trained with 256 input neurons and 15 output neurons (one for each symbol). We have achieved 84.34% of recognition accuracy.

The summary of all the three zone symbols of the Gujarati script is depicted in the table 2.

Table 2 classification results

Sr. No.	Zones	Number of symbols	Accuracy
1	Upper zone	11	84%
2	Middle zone	119	97.59%
3	Lower zone	4	97%

These results demonstrate the highest level of accuracy in all the three zones and too in a very short time.

5.6. Conclusions

The General Regression Neural Network (GRNN), based on sound statistical techniques discussed in sections 5.3 and 5.4, is one time run approach. Therefore, the network need not undergo the training which is the time consuming process of learning. That is the biggest advantage of GRNN over the traditional ANN architectures. Results of recognition of the symbols of all the three zones of Gujarati script viz. lower, middle and upper are found to be very encouraging as shown in table-5.2 of section 5.5. These recognition accuracies are higher than those quoted in the literature for Gujarati script. By identifying the similar looking characters, the confusion set is generated [30] which can be classified by sub glyph level analysis at the time of post processing.