Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2015, Article ID 193868, 12 pages
http://dx.doi.org/10.1155/2015/193868

Research Article

On the Performance Improvement of Devanagari Handwritten Character Recognition

Pratibha Singh,[1] Ajay Verma,[1] and Narendra S. Chaudhari[2]

[1] *IET, DAVV, Khandwa Road, Indore 452017, India*
[2] *IIT, Khandwa Road, Indore 452017, India*

Correspondence should be addressed to Pratibha Singh; prat_ibh_a@yahoo.com

Received 3 August 2014; Revised 29 December 2014; Accepted 16 January 2015

Academic Editor: Ying-Tung Hsiao

The paper is about the application of mini minibatch stochastic gradient descent (SGD) based learning applied to Multilayer Perceptron in the domain of isolated Devanagari handwritten character/numeral recognition. This technique reduces the variance in the estimate of the gradient and often makes better use of the hierarchical memory organization in modern computers. $L2$-weight decay is added on minibatch SGD to avoid overfitting. The experiments are conducted firstly on the direct pixel intensity values as features. After that, the experiments are performed on the proposed flexible zone based gradient feature extraction algorithm. The results are promising on most of the standard dataset of Devanagari characters/numerals.

1. Introduction

The need to recognize the handwritten text is challenging problem not only from the perspective of behavioural biometrics but also in the context of pattern recognition. Writing is the most natural mode of collecting, storing, and transmitting the information. It is a widely used communication tool among human being and forms the input for simulation of reading by a machine. The intensive research effort in the field of character recognition (CR) was due to challenges on simulation of human reading and also because of its potential applications, for example, postal automation, bank cheque analysis and processing, conversion of handwritten text into Braille, hand drawn pictogram or formula recognition, and so forth. Pattern recognition is a computationally intensive and time-consuming task due to vast amount of image data and large number of computational steps involved. The great demand for fast classification of letters by the post office requires a fast automated recognition system. Traditionally, the conventional approach always demands a very high speed computer or a parallel computer system to perform a satisfactory and fast recognition. We cannot meet these demands using simple digital computer. Digital computers are good at handling problems which are explicitly formulated, but handwritten character recognition is not such a problem. With the advent of neurocomputing technology, the great research effort has been devoted to formulate the pattern recognition tasks in an efficient manner. Present study investigates the direction for the improvement of performance in Devanagari CR system.

There are 18 official languages accepted in the present Indian constitution. Twelve different scripts are used for writing these languages. Many of the Indian documents are supposed to be written in three languages, namely, English, Hindi, and the state official language as per the three language formula [1]. Hindi is the popularly used language of India and is the third most popular language in the world, which is written and encoded using Devanagari script. Not only Hindi but also the other languages such as Marathi, Sanskrit, and Konkani are encoded into Devanagari script. Basic characters in Devanagri script consists of 13 vowels and 36 consonants [2] as shown in Figure 1. Writing style is from left to write. There is no concept of upper and lower case. Vowels following consonants take a modified shape and are known as modified characters. Shape of modified characters varies depending on whether the vowel modifier is placed to the left, right, top, or bottom of the consonants as shown in Table 1. In Devanagari script, there is a practice of using more than twelve different forms each of 36 consonants [3], giving rise to its shape variation. The existence of modifiers and

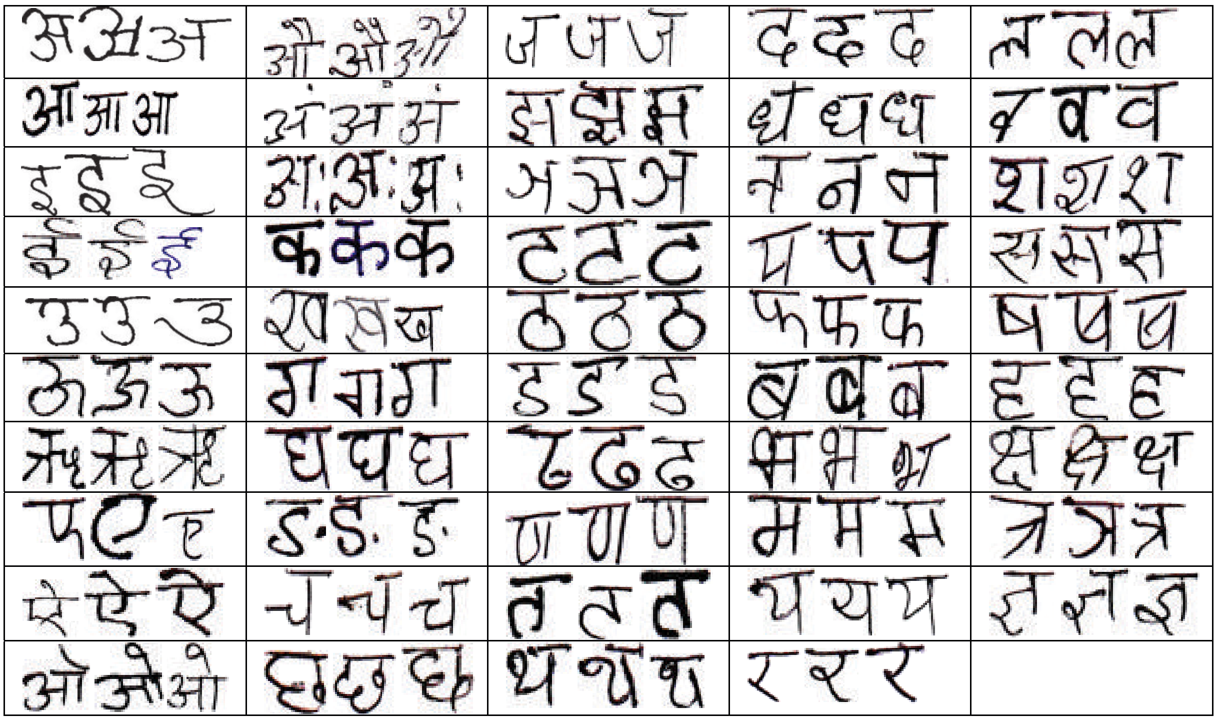


Figure 1: Basic characters of Devanagari script.

Table 1: Modifiers in Devanagari script.

Vowel	आ	उ	ऊ	इ	ई	ऋ	ए	ऐ	ओ	औ	अं	अः
Modifier	ा	ु	ू	ि	ी	ृ	े	ै	ो	ौ	ं	ः
Modified shape of क	का	कु	कू	कि	की	कृ	के	कै	को	कौ	कं	कः

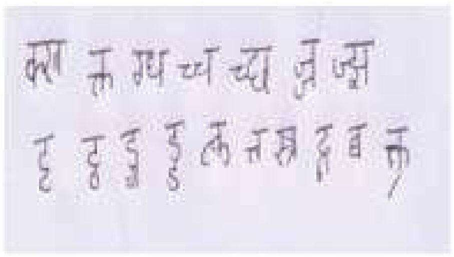


Figure 2: Compound characters of Devanagari script.

the compound characters, shown in Figure 2, makes the character recognition more difficult for Devanagari script.

A key reason for the absence of sustained research efforts in Devanagari Optical Character Recognition (OCR) is mainly because of the paucity of data resources. Ground-truthed data for words and characters, on-line dictionaries, corpora of text documents, reliable standardized statistical analysis, and evaluation tools are currently lacking. So, the creation of such data resources will undoubtedly provide a much needed fillip to researchers working on Devanagari OCR. The major research for isolated Devanagari characters is done by Computer Vision and Pattern Recognition (CVPR) Unit of Indian Statistical Institute (ISI), Kolkata. Fuzzy model based recognition scheme was proposed by Hanmandlu and Murthy [4] for isolated Devanagari numerals. Classifier combination is now widely applied on Devanagari CR system for increasing recognition accuracy [5–9]. Classifier combination techniques using shadow features is proposed by Arora et al. [3, 10]. MLP and HMM combination schemes have been proposed by Bhattacharya et al. [6].

Pattern recognition applications use the algorithm of machine learning. Machine learning in supervised classification domain mainly involves two steps: training and testing. One of the way to improve the performance of machine learning algorithm is to use a low bias algorithm and to train the algorithm with a large data typically known as big data. But learning with large dataset comes with its own computational problems and requires a million aggregations over each step. So, massive computation cost in the most popular algorithms such as gradient descent requires alternative solution. The present study focuses on this issue for the larger dataset of Devanagri. Stochastic gradient descent, online learning, and minibatch learning are some of the alternatives to deal with this issue. The rest of the paper is organized as follows: Section 2 talks about the methodology used, Section 3 gives the description of classifier model used, Section 4 describes

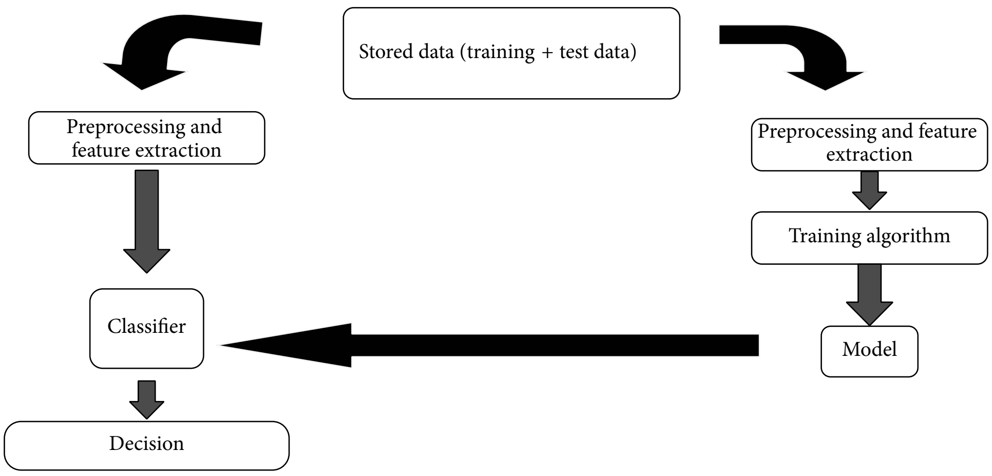

Stored data (training + test data)
Preprocessing and feature extraction
Classifier
Decision
Preprocessing and feature extraction
Training algorithm
Model

FIGURE 3: Design cycle of handwriting recognition system.

the evaluation results on the Devanagari datasets, and final concluding remarks are presented in Section 5.

2. Methodology Used

The design cycle for the recognition of characters follows all the steps of standard pattern recognition technique in supervised learning mode. For the purpose of training, datasets developed by various research groups are used. To the best of our knowledge, only isolated characters and numeral dataset of Devanagari script is available as test bed. So, these are experimented in this study. The design cycle used in this study is shown in Figure 3.

The images obtained from benchmarking dataset exist in two groups, namely, training and testing. The images are subjected to various preprocessing steps described in Table 2. The output after preprocessing module is not suitable for the classifier training because of the high dimensionality. Feature extraction/selection is an important step for the dimensionality reduction. In this study, we have used a gradient based directional features. The features are obtained from the 9 different parts of the image sample. The global and local histogram based zone boundary concept [11] as shown in Figure 4, is used before feature extraction.

2.1. The Feature Extraction Module. The features are accumulated zonewise using two different alternatives of zoning. For example,

(i) standard zoning: the entire bounding box of the image is divided into 3 × 3 zones, and gradients are accumulated for each zone;

(ii) elastic zoning: the concept of elastic zoning is based on equalizing the density of each zone. We here define global or local zoning concept. In global zoning, the zone separating line is decided on the basis of equal density division, horizontally and vertically, whereas in local method the image is divided horizontally based on density equalization in each zone and then the vertical boundary is decided on the local division of density.

2.1.1. The Gradient Features. The gradient feature decomposition was originally proposed for online character recognition. This method is applicable to machine printed/handwritten and binary/grayscale, as well as low resolution images. Conventionally, the gradient is calculated on each pixel of the image. In our analysis, we have applied "Sobel" edge detection algorithm to calculate gradient vector at each image pixel of preprocessed image. The gradient vector can be quantized into eight directions using two methods, namely, angle vector quantization and vector decomposition using parallelogram rule. In the first method, the magnitude of gradient in each image pixel is assigned to its nearest directional plane and in the second method the gradient vector is decomposed into two nearest directional planes using parallelogram vector division rule. The parallelogram quantization method gives less quantization error, so we have taken this method for quantizing gradient vector.

The calculated gradient of the image is decomposed into four, eight, or sixteen directional planes. For our analysis, we have taken eight directional planes. Figure 5(c) shows the gradient vector decomposition into their nearest vector plane. We have accumulated the magnitude of gradient in eight discrete directions for each of the subsection of original image. The components of gradient vector are given by the following equations:

$$\begin{aligned} gx(x,y) &= f(x+1,y-1) + 2f(x+1,y) \\ &\quad + f(x+1,y+1) - f(x-1,y-1) \\ &\quad - 2f(x-1,y) - f(x-1,y+1), \\ gy(x,y) &= f(x-1,y+1) + 2f(x,y+1) \\ &\quad + f(x+1,y+1) - f(x-1,y-1) \\ &\quad - 2f(x,y-1) - f(x+1,y-1). \end{aligned} \tag{1}$$

TABLE 2: Proposed feature extraction algorithm.

Input: image (training/test)	Output: features of image
Preprocessing steps	(1) Convert the image into two-tone image (2) Normalize the image to standard size (3) Obtain the boundary image (4) Obtain the features using algorithms 1 or 2 or 3
Algorithm 1 (standard zone)	For boundary image IM(n, m) (1) divide the n and m by number of bins in horizontal and vertical directions to calculate the value of boundary indices u_i and v_i (2) obtain the subimages IM$_i$ generated from IM using u_i and v_i (3) go for feature extraction of each subimage IM$_i$
Algorithm 2: (global zone)	For boundary image IM(n, m) (1) calculate density in horizontal direction D_H (2) calculate density in vertical direction D_V (3) divide the density D_H/D_V by number of bins in horizontal/vertical direction to calculate D_h/D_v; obtain the indices u_i/v_i for zone boundary using D_h/D_v (4) obtain the subimages IM$_i$ generated from IM using u_i and v_i (5) go for feature extraction of each subimage IM$_i$
Algorithm 3: (local zone)	For boundary image IM(n, m) (1) calculate density in horizontal direction D_H (2) calculate density in vertical direction D_V (3) divide the density D_H and D_V by number of bins in horizontal and vertical directions to calculate density for locating the boundary indices of zone u_i and v_i (4) obtain the subimages IM$_i$ generated from IM using only u_i (5) obtain for each IM$_i$ density D_{Hh} and divide this by the number of bins in horizontal direction to calculate density for locating vertical index of zone w_i (6) obtain subimage from IM$_j$ using u_i and w_i (7) go for feature extraction of each subimage IM$_j$

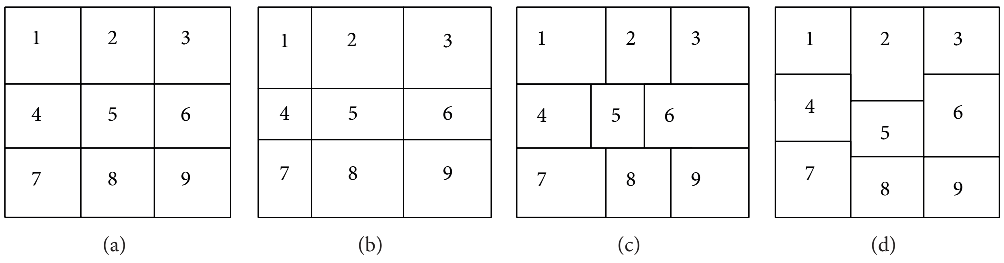


FIGURE 4: (a) Standard zoning; (b) global zoning; (c) and (d) local zoning.

3. The Classifier Model

Multilayer Perceptron is used as classifier. The architecture of Multilayer Perceptron (MLP) consists of input layer, output layer, and hidden layer. Single hidden layer perceptron gives universal approximation in many pattern recognition applications. The output vector for a single layer perceptron is given by

$$f(x) = G\left(b^{(2)} + W^{(2)}\left(s\left(b^{(1)} + W^{(1)}x\right)\right)\right), \tag{2}$$

where $b^{(1)}$, $b^{(2)}$ are the bias vectors at the hidden and output layers, $W^{(1)}$, $W^{(2)}$ are the weight matrices at the respective nodes, and s, G are the activation functions. For a classification problem, if $(x^{(i)}, y^{(i)})$ is the training vector, where $x^{(i)} \in \Re^D$, a D-dimensional training vector, and $y^{(i)} \in \{1, \ldots, L\}$. For the prediction function $f(x)$ given in (2), the zero-one loss function is given by

$$\ell_{0,1} = \sum_{i=0}^{|\mathcal{D}|} I_{f(x^{(i)} \neq y^{(i)})}, \tag{3}$$

where I is the indicator function given by

$$I_x = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise,} \end{cases}$$
$$f(x) = \underset{k}{\operatorname{argmax}} P(Y = k \mid x, \theta), \tag{4}$$

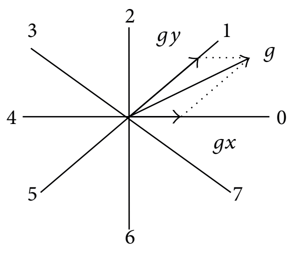
where θ is the set of all parameters of the given model. The objective of the training is to minimize the loss function. But, the zero-one loss function is not differentiable; therefore,

−1	0	1
−2	0	2
−1	0	1

(a)

−1	−2	−1
0	0	0
1	2	1

(b)



(c)

Figure 5: (a) Sobel mask for vertical gradient; (b) Sobel mask for horizontal gradient; (c) quantization using parallelogram rule.

negative log likelihood of loss function minimization is used as the objective of the training

$$\mathrm{NLL}(\theta, \mathscr{D}) = -\sum_{i=0}^{|\mathscr{D}|} P\left(Y = y^{(i)} \mid x^{(i)}, \theta\right). \tag{5}$$

Weights are updated using gradient of the error surface defined by loss function. Gradient is estimated from the training data. In this study, stochastic gradient descent based learning (Table 3) approach is applied to MLP. In ordinary gradient descent algorithm, repeatedly small steps are made in downward direction on an error surface, which is the mean square error. Mean square error is a function of weights. Stochastic gradient descent (SGD) works according to the same principles as ordinary gradient descent, but it proceeds more quickly by estimating the gradient from just a few examples at a time instead of the entire training set. In its purest form, estimation of the gradient is made from just a single example at a time. In both gradient descent (GD) and stochastic gradient descent (SGD), we update a set of weights in an iterative manner to minimize an error function. In normal GD (Table 4), all the samples of the training set have to be processed before updating the weight for a particular iteration, while, in SGD, *only single* training sample from whole training set is used to do the update for a weight in a particular iteration. Thus, for big data, if the number of training samples is very large, then using gradient descent may take too long because in every iteration, when we are updating the values of the parameters, we are running through the complete training set. On the other hand, using SGD will be faster because you use only one training sample and it starts improving itself right away from the first sample. SGD often converges much faster compared to GD, but the error function is not as well minimized as in the case of GD. Often, in most cases, the close approximation that we get in SGD for the parameter values is enough because they reach the optimal values and keep oscillating there. Stochastic gradient descent has a convergence rate which is independent of the size of our dataset and is thus adapted when we have a huge or even infinite dataset. But, there are two downsides to it:

(i) its slow convergence rate: it is not beneficial to get a faster convergence rate on the training set as this will not be translated into a better convergence rate on the test set [12];

Table 3: Stochastic gradient descent.

```
For ($x_i$,$y_i$) in training set,
% assume an infinite generator
% It may repeat examples (if there is only a finite training loss)
loss = $f$(parameters, $x_i$, $y_i$)
find derivative of loss with respect to parameters % compute gradient
modify parameters by-learning rate $*$ derivative of loss with respect to parameters
if ⟨stopping condition is met⟩,
return parameters
```

Table 4: Gradient descent.

```
While being true,
loss = $f$(parameters)
find derivative of loss with respect to parameters % compute gradient
modify parameters by-learning rate $*$ derivative of loss with respect to parameters
if ⟨stopping condition is met⟩
return parameters
```

Table 5: Minibatch-SGD.

```
for ($x_{\text{batch}}$, $y_{\text{batch}}$) in training batches,
% assume an infinite generator
% it may repeat examples
loss = $f$(parameters, $x_{\text{batch}}$, $y_{\text{batch}}$)
find derivative of loss with respect to parameters % compute gradient
modify parameters by-learning rate $*$ derivative of loss with respect to parameters
if ⟨stopping condition is met⟩,
return parameters
```

(ii) its sensitivity to the two parameters, the learning rate, and the decrease constant.

For deep learning, a variant that we recommend is a further modification in stochastic gradient descent using the so-called "minibatches." Minibatch SGD (MSGD), explained in Table 5, works identically to SGD, except that we use more

Table 6: Number of image samples in CPAR-12 numeral dataset.

Image	0	1	2	3	4	5	7	8	9^1	9^2
Train dataset	2280	2280	2280	2280	2280	2280	2280	2280	2280	1200
Test dataset	1012	1012	1012	1012	1012	1012	1012	1012	1012	880
Total	3292	3292	3292	3292	3292	3292	3292	3992	3292	2080

There are two forms of writing digit "9" in Devanagari, ^{1}for form 1 while ^{2}for form 2.

than one training example to make each estimate of the gradient. This technique reduces variance in the estimate of the gradient and often makes better use of the hierarchical memory organization in modern computers.

There is a trade-off in the choice of the minibatch size B. With large B, time is wasted in reducing the variance of the gradient estimator, that time would be better spent on additional gradient steps. An optimal B is model, dataset, and hardware dependent and can be anywhere from 1 to several hundreds. In machine learning, when we train our model from data, we are trying to prepare it to do well on new examples, not the ones it has already seen. The MSGD training loop does not have the capability for this generalization and have a tendency for overfitting. The way to combat overfitting is through regularization and early stopping using validation. There are several techniques for regularization; the most commonly used method is $L1/L2$ regularization which is explained in the next section.

3.1. Weight Decay. Weight decay is a subset of regularization methods. The penalty term in weight decay, by definition, penalizes large weights. Other regularization methods may involve not only the weights but also various derivatives of the output function. The weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would. Large weights can hurt generalization in two different ways. Excessively large weights leading to hidden units can cause the output function to be too rough, possibly with near discontinuities. Excessively large weights leading to output units can cause wild outputs far beyond the range of the data if the output activation function is not bounded to the same range as the data. To put it another way, large weights can cause excessive variance of the output.

3.2. L1 and L2 Regularization. $L1$ and $L2$ regularization involve adding an extra term to the loss function, which penalizes certain parameter configurations. Formally, if our negative log likelihood loss function is $\mathrm{NLL}(\theta, |\mathfrak{D}|)$, then the regularized loss will be given by

$$E(\theta, \mathfrak{D}) = \mathrm{NLL}(\theta, |\mathfrak{D}|) + \lambda R(\theta). \tag{6}$$

This is written as follow for the present study:

$$E(\theta, \mathfrak{D}) = \mathrm{NLL}(\theta, |\mathfrak{D}|) + \lambda \|\theta\|_p^p, \tag{7}$$

where $\|\theta\|_p$ is the L_p norm of θ

$$\|\theta\|_p = \left(\sum_{j=0}^{|\theta|} \left|\theta_j\right|^p\right)^{1/p}, \tag{8}$$

λ is a hyperparameter which controls the relative importance of the regularization parameter. Commonly used values for p are 1 and 2, hence the $L1/L2$ nomenclature. If $p = 2$, then the regularizer is also called "weight decay." In principle, adding a regularization term to the loss will encourage smooth network mappings in a neural network (by penalizing large values of the parameters, which decreases the amount of nonlinearity that the network models). More intuitively, the two terms (NLL and $R(\theta)$) correspond to modeling the data well (NLL), having "simple" or "smooth" solutions. Thus, minimizing the sum of both will, in theory, correspond to finding the right trade-off between the fit to the training data and the "generality" of the solution that is found.

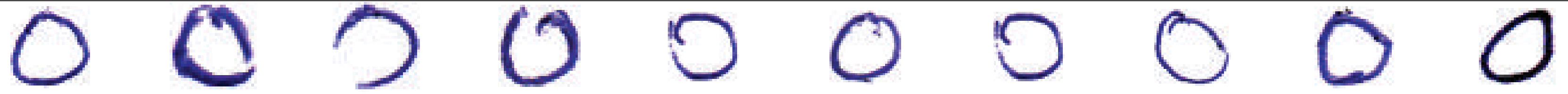
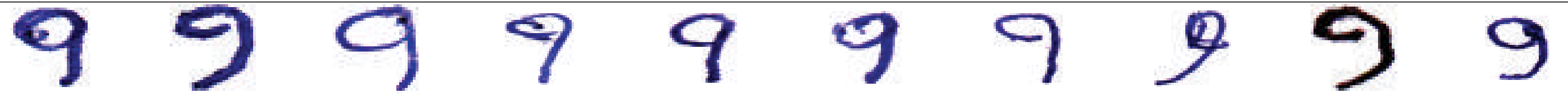
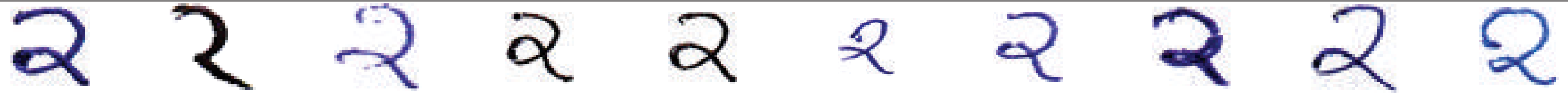
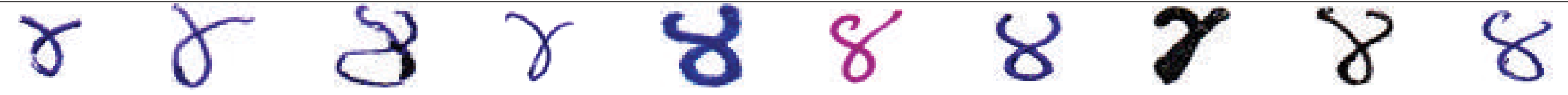
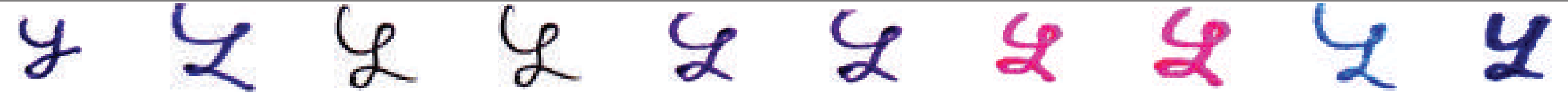
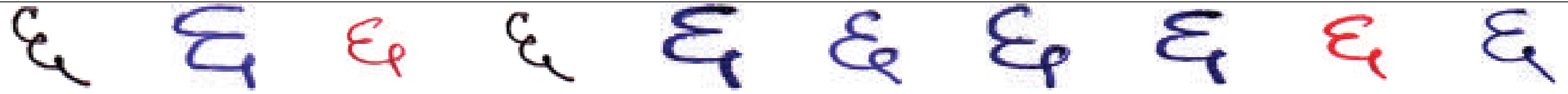
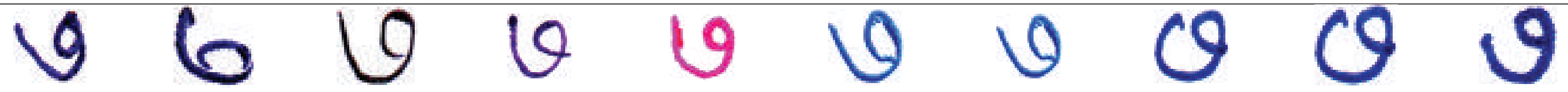
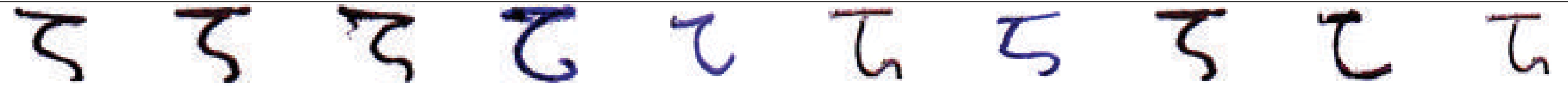
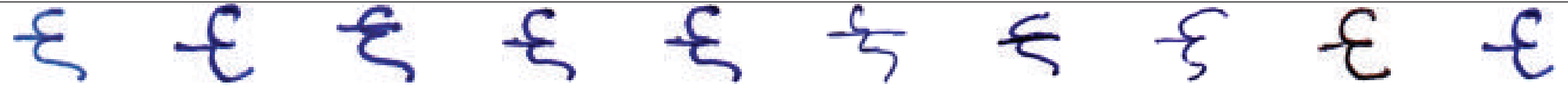
4. Experimental Results

The experiments are performed on various datasets as described in the next subsections. The images from the dataset (training and testing) are preprocessed. The feature extraction method as discussed in Table 2 is applied and Feed-Forward Neural Network based classifier is used which is incorporated with $L2$-Regularization and minibatch. For implementing the Feed-Forward Neural Network using minibatch approach, the code developed by Palm [13] is used in MATLAB. The neural network configuration as specified in column 4 of Table 9 is either a three layer or a four layer network. The first value of the configuration specifies the number of features, last value specifies the number of classes and the rest (between input and output) specifies the units present in the hidden layer. The experiments are performed on Intel Core i-3 processor with 6 GB RAM. In next subsection, the information about various datasets is given.

4.1. The MNIST Dataset. The MNIST [14] dataset consists of handwritten digit images and it is divided in 60,000 examples for the training set and 10,000 examples for testing. All digit images have been size-normalized and centered in a fixed size image of 28 × 28 pixels. In the original dataset, each pixel of the image is represented by a value between 0 and 255, where 0 is black, 255 is white, and anything in between is a different shade of grey. An image is represented as 1-dimensional array of 784 (28 × 28) float values between 0 and 1 (0 stands for black and 1 for white). The labels are numbers between 0 and 9 indicating which digit the image represents. When using the dataset, we usually divide it in minibatches.

4.2. The CPAR-2012 Dataset. This dataset is available since the year 2012 [15] to the research community and is developed

Table 7: Image samples of CPAR-2012 handwritten numerals [15].

Zero	
One	
Two	
Three	
Four	
Five	
Six	
Seven	
Eight	
Nine[1]	
Nine[2]	

There are two forms of writing digit "9" in Devanagari, [1]for form 1 while [2]for form 2.

Table 8: Sample images from CVPR Dataset [16].

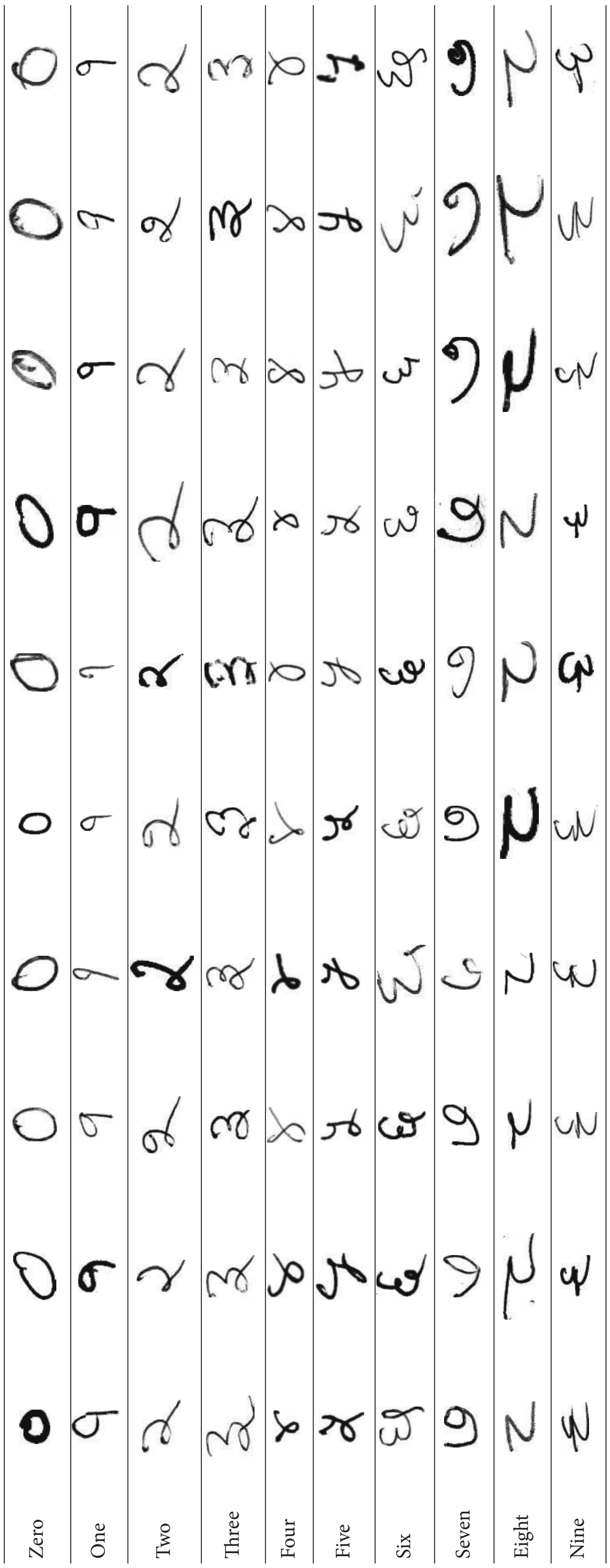

Zero
One
Two
Three
Four
Five
Six
Seven
Eight
Nine

Table 9: Error rates for various datasets using direct pixel values.

Dataset	Training samples	Test samples	Neural net configuration	Time for recognition of dataset including training	Percentage error
MNIST handwritten digit	50000	10000	784-100-10	4846 sec	2.25
MNIST handwritten digit	50000	10000	784-200-100-10	9471 sec	2.15
MNIST Handwritten digit	50000	10000	784-200-10	7923 sec	2.13
ISI digit	18000	3500	784-100-10	1524 sec	3.31
ISI digit	18000	3500	784-200-100-10	3239 sec	3.17
ISI digit	18000	3500	784-200-10	2767 sec	2.74
CPAR-2012 character	49000	29400	784-100-49	3903 sec	21.54
CPAR-2012 character	49000	29400	784-200-49	8334 sec	18.6
CPAR-2012 character	49000	29400	784-200-100-49	8794 sec	17.21
CPAR-2012 numeral	26250	8750	784-200-11	5422 sec	2.53
CPAR-2012 numeral	26250	8750	72-100-11	2153 sec	2.8
CPAR-2012 numeral	26250	8750	784-200-100-11	5560 sec	2.77

Table 10: Error rates for various datasets using proposed feature extraction method.

Dataset	Feature extraction method	Number of features	NN configuration	Time for training and testing	Error rate
CPAR-2012 character	Global zone based edge	72	72-200-49	1941 sec	14.89
CPAR-2012 character	local zone based edge	72	72-200-49	1864 sec	16.01
CPAR-2012 character	Equal zone	72	72-200-49	2082 sec	21.35
ISI digit	Equal zone edge	72	72-200-10	1224 sec	2.03
ISI digit	Global zone based edge	72	72-200-10	1178 sec	1.83
ISI digit	local zone based Edge	72	72-200-10	909 sec	2.14
CPAR-2012 numeral	Global zone based edge	72	72-200-10	756 sec	2.38
CPAR-2012 numeral	local zone based edge	72	72-200-10	785 sec	1.93
CPAR-2012 numeral	Equal zone edge	72	72-200-10	956 sec	2.07

by Intelligent System Group Noida (Centre for Pattern Analysis and Recognition, CPAR). This is the largest dataset available for the handwritten isolated patterns. It consists of 35000 images of numerals and 78400 images of characters. The data is collected from diverse population strata of 2000 writers from various states of India having different religions. The numeral dataset is having 11 classes in this dataset because the pattern corresponding to number "9" can be written in two alternate ways as shown in Table 7. The character dataset is having 1000 training images and 600 test images of 49 classes each. Table 6 describes the number of pattern for each numeral class of CPAR-12 used in this study.

4.3. CVPR-ISI Dataset. This dataset is available for the global research community since the year 2009 and is developed by CVPR unit of ISI Kolkata. The Devanagari numeral database includes samples collected from mail pieces and job application forms through specially designed form for data collection. The dataset consists of 22,556 images (as shown in Table 8) stored in "tif" format, collected from 1049 writers.

In this study, experiments are performed with single hidden layer (having 200 or 100 hidden units) NN and two hidden layer NN (having 200 and 100 hidden units resp.). Input layer is made up of the number of units equal to the size of feature vector generated. Network is trained for 200 epoch with a mini batch size of 100. The learning rate used in the experiments is kept constant to be equal to 1. The value of regularization weight decay $L2$ is maintained at $1e-04$. The activation function used in hidden layer is hyperbolic tangent and in the output layer it is logistic. The momentum set for experiments is equal to 0.7. The error rates obtained for the various dataset using direct pixel values as features are tabulated in Table 9. The results are obtained for the size normalized image patterns with the features as direct pixel intensities. The network configuration varies with respect to the number of hidden layers. The time for recognition includes the training time along with the classification.

Table 11: Comparison of performance of proposed method with the previously reported results.

Dataset	Approach used by previous reported results	Feature used by previous reported results	Classifier used by previous reported results	Previously reported recognition percentage	Recognition percentage of proposed method (direct pixel as features)	Recognition percentage of proposed method (gradient feature)
CPAR-2012 digit [17]	Classifier combination (MV)	Direct pixel + profile + gradient + wavelet transform	CCN, FNN, PRN, KNN, FFT	97.87% (35000)	97.47 (35000)	98.07 (35000)
CPAR-2012 character [9]	Classifier combination (MV)	Direct pixel + profile + Gradient + wavelet transform	CCN, FNN, PRN, KNN, FFT	84.03% (79400)	82.79% (79400)	**85.11 (79400)**
ISI devanagari digit [18]	Feature combination	PCA/MPCA + QTLR	SVM	98.55% **(3000) CMATER data**	**97.26 (22546) Full ISI Data**	**98.17 (22546) Full ISI Data**
ISI devanagari digit [16]	Multistage classifier	Wavelet	Multistage MLPs	99.04 with 0.24% rejection		
ISI devanagari digit [7]	Ensemble using AdaBoost	Zernike moments	MLPs	96.80 (single) **(22546)**		

Abbreviations used in Table 11 are as follows: PCA: principal component analysis, KNN: K-nearest neighbor, FNN: feed-forward neural network, SVM: support vector machine, MLP: multilayer perceptron, CNN: cascade neural network, PRN: pattern recognition network, and FFT: function fitting neural network.

Table 10 illustrates the performance of various feature extraction algorithms described in Table 2 on SGD based learning of MLP. All the sample images are partitioned into 3×3 subimages, but the criterion for zoning is different. Edge based directional features are extracted from each of the subimage in 8 discrete directions defining 72D feature vector in each case. Three layer architecture of MLP is used with 200 nodes in hidden layer. The rest of the NN-settings are kept as same as in the previous experiment on direct pixel values. The number of data samples used in this case for training and testing is same as that mentioned in Table 9 (column 2 and column 3).

4.4. Performance Comparison for Numeral Dataset CPAR-2012. In the experiments performed by Kumar and Ravulakollu [17], the average recognition rate reported was 95.18% with rejection of some of the samples, on a single classifier and 97.87% with rejection of some of samples on an ensemble of classifier is applied to the patterns. For the same dataset, the proposed learning strategy gives the recognition rate of 98.07% without any rejection. *For the CPAR-2012 numeral dataset, the accuracy the proposed features with SGD learning of MLP is improved by 0.2%*. Here, the point to be emphasized is that our results are better than the previous results because of two reasons: (1) we have not used any rejection of samples and (2) we have not used model of classifier ensemble. The recognition results reported here in this work are for the single classifier.

4.5. Performance Comparison for Character Dataset CPAR-2012. Kumar and Ravulakollu [17] applied the method of classifier ensemble on the character dataset and obtained the recognition rate of 84.03% for a rejection of 5.3%. For the same dataset, the proposed learning method yields the recognition rate of 82.79% with direct pixel values and 85.11% with feature extraction method without any rejection of patterns. *For CPAR-2012 character dataset, the accuracy improvement of 1.08% is observed with the proposed feature set along with the SGD learning on MLP*. Our results are better from the point of view of the reasoning given in the previous section.

4.6. Performance Comparison for Numeral Dataset ISI-CVPR. For ISI Devanagari numerals performance comparison is made with the three previously reported results. The result reported by Das et al. [18] is better, but it is tested for small subset of ISI data, whereas our method is tested on complete ISI dataset. Result reported by Bhattacharya and Chaudhuri [16] is tested on complete dataset, but the accuracy is obtained with *0.24% rejection* of patterns and features used in the study belong to complex algorithms of feature extraction. Also, the classification algorithm used by them is a multistage MLP based classifier and there is a rejection of patterns used by the last stage in multistage MLP. For the single k-NN (K-nearest neighbor), the benchmark established by them is of accuracy level 97.26%. So, for ISI data considering the complexity of the algorithm, our results are better. The proposed method is better compared with the method developed by Bhattacharya and Chaudhuri [16] in terms for speed of recognition.

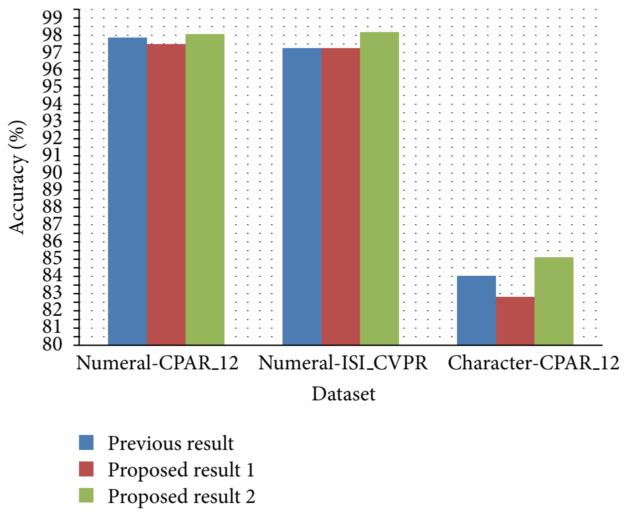


FIGURE 6: Performance of the proposed method with the previously reported result.

5. Conclusion

The minibatch stochastic gradient is used to accelerate the speed of recognition for large dataset. The recognition results are obtained using direct pixel intensities as feature on MSGD. Secondly, the results are obtained for edge based directional features on MSGD. From the results it is clear that if Pre-processing and feature extraction is used along with the mini batch algorithm, the error rate reduces by a 1–3% over direct pixel intensity features. The proposed method gives better/same recognition accuracy with most of the standard benchmarks available for Devanagari characters. The recognition time cannot be compared from the previously reported results as the time was not considered as criterion in previous research. The proposed method is faster over normal gradient descent based learning and it gives good accuracy on even direct pixel intensities. The performance improvement of the proposed methods is given in terms of accuracy as tabulated in column 6 and 7 of Table 11. In this table the first column is giving the information about the dataset and the reference associated with the previous result. For showing the effectiveness of our proposed strategy we have taken the same dataset of Devanagari characters. Column 2 is summarizing the approach used by the previously reported results, column 3 provides features used by them, column 4 gives the classification method used by the previous research and column 5 of Table 11 provides the accuracy of previously reported results by other researchers.

Figure 6 is showing the accuracy improvement of the proposed results over the existing methods.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank Intelligent System Group Noida and CVPR Unit of ISI Kolkata for providing the dataset of Devanagari characters/numerals.

References

[1] Department of Official Language, 1968, http://www.rajbhasha.nic.in/GOLPContent.aspx?t=endolresolution.

[2] N. Sharma, U. Pal, F. Kimura, and S. Pal, "Recognition of off-line handwritten devnagari characters using quadratic classifier," in *Computer Vision, Graphics and Image Processing : Proceedings of the 5th Indian Conference, ICVGIP 2006, Madurai, India, December 13-16, 2006*, vol. 4338 of *Lecture Notes in Computer Science*, pp. 805–816, Springer, Berlin, Germany, 2006.

[3] S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu, "Recognition of non-compound handwritten Devanagari characters using a combination of MLP and minimum edit distance," *International Journal of Computer Science and Security*, vol. 4, no. 1, pp. 1–14, 2010.

[4] M. Hanmandlu and O. V. R. Murthy, "Fuzzy model based recognition of handwritten numerals," *Pattern Recognition*, vol. 40, no. 6, pp. 1840–1854, 2007.

[5] S. Shelke and S. Apte, "A novel multi-feature multi-classifier scheme for unconstrained handwritten devanagari character recognition," in *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition (ICFHR '10)*, pp. 215–219, kolkata, India, November 2010.

[6] U. Bhattacharya, S. K. Parui, B. Shaw, and K. Bhattacharya, "Neural combination of ANN and HMM for handwritten Devnagari numeral recognition," in *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*, pp. 613–618, La Baule, France, 2006.

[7] T. Jindal and U. Bhattacharya, "Recognition of offline handwritten numerals using an ensemble of MLPs combined by adaboost ," in *Proceedings of the 4th International Workshop on Multilingual OCR (MOCR '13)*, vol. 18, Washington, DC , USA, August 2013.

[8] S. Kumar, "A three tier scheme for devanagari hand-printed character recognition," in *Proceedings of the World Congress on Nature and Biologically Inspired Computing (NABIC '09)*, pp. 1016–1021, Coimbatore, India, December 2009.

[9] R. Kumar and K. K. Ravulakollu, "On the performance of Devnagari handwritten character recognition," *World Applied Sciences Journal*, vol. 31, no. 6, pp. 1012–1019, 2014.

[10] S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu, "Combining multiple feature extraction techniques for Handwritten Devnagari Character recognition," in *Proceedings of the 3rd International Conference on Industrial and Information Systems (ICIIS '08)*, pp. 1–6, IEEE, Kharagpur, India, December 2008.

[11] P. Singh, A. Verma, and N. S. Chaudhari, "Performance analysis of flexible zone based features to classify Hindi numerals," in *Proceedings of the 3rd International Conference on Electronics Computer Technology (ICECT '11)*, pp. 292–296, Kanyakumari, India, April 2011.

[12] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Optimization for Machine Learning*, S. Sra, N. Sebastian, and W. Ste, Eds., pp. 351–368, MIT Press, Cambridge, Mass, USA, 2011.

[13] R. B. Palm, *Prediction as a candidate for learning deep hierarchical models of data [M.S. thesis]*, 2012.

[14] Y. LeCun, C. Cortes, and C. J. C. Burges, *The MNIST Dataset of Handwritten Digits*, 1998, http://yann.lecun.com/exdb/mnist/.

[15] R. Kumar, A. Kumar, and P. Ahmed, *A Benchmark Dataset for Devanagari Document Recognition Research*, WSEAS Press, Lemesos, Cyprus, 2013.

[16] U. Bhattacharya and B. B. Chaudhuri, "Handwritten numeral databases of Indian scripts and multistage recognition of mixed numerals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 444–457, 2009.

[17] R. Kumar and K. K. Ravulakollu, "Handwritten devnagari digit recognition: benchmarking on new dataset," *Journal of Theoretical and Applied Information Technology*, vol. 60, no. 3, pp. 543–555, 2014.

[18] N. Das, J. M. Reddy, R. Sarkar et al., "A statistical-topological feature combination for recognition of handwritten numerals," *Applied Soft Computing Journal*, vol. 12, no. 8, pp. 2486–2495, 2012.