Devanagari Character Recognition towards natural Human-Computer Interaction

Pulkit Goyal
IIIT Allahabad, Amethi Campus
Teekarmaphi, Amethi
Dist. Sultanpur(U.P) - 227413
pulkit110@gmail.com

Sapan Diwakar
IIIT Allahabad, Amethi Campus
Teekarmaphi, Amethi
Dist. Sultanpur(U.P) - 227413
diwakar.sapan@gmail.com

Anupam Agrawal
IIIT Allahabad
Deoghat, Jhalwa
Allahabad (U.P.) - 211012
anupam@iiita.ac.in

Human-computer interaction is a growing research area. There are several ways of interaction with the computer. Handwriting has continued to persist as a means of communication and recording information in the day to day life even with the introduction of new technologies. Due to the growth of technology in India, it becomes important to devise ways that allow people to communicate with computer in Indian languages. Hindi being the national language of India, we present a way to communicate with the computer in Hindi or more precisely, 'Devanagari script'. Due to absence of a global font to represent Devanagari characters, it is important that the computer recognizes the characters written by the user in order to interact with him. The algorithm implemented for character recognition first segments the image containing Devanagari text fed to the software into lines, lines to words and words to characters. The obtained characters are then brought down to a standard size. The Kohonen Neural Network based recognizer then comes into action and recognizes the text character by character and provides the output in Unicode format. The network has been designed with no hidden layer to support quick recognition. Apart from text recognition from an image, we also provided the option to recognize individual handwritten characters drawn using a mouse. Such a system provides keyboard less computer interaction. The technique is implemented using Java. The overall recognition rate for a fixed font machine printed characters is 90.26% and for hand written characters, it is 83.33%.

Handwriting Recognition, Segmentation, Kohonen Neural Network, Self Organizing Map, Devanagari Characters

1. INTRODUCTION

Hindi is used by more than 400 million people across the globe [6]. It consists of eleven vowels and thirty three consonants giving a total of forty four characters. Every word in Devanagari script is written by first drawing a horizontal line which is called *Shirorekha* and then writing the characters beneath *Shirorekha.* Characters can be joined with other characters and with vowels as well.

Even with the advancing technology in India, there is a lack of such software which can recognize Devanagari text. Such software has many applications. It may be used in communicating with the computer in our native language by sketching characters. It can also be useful for indexing images for search engines. This is necessary because most of the websites use images to represent Devanagari text. It can also be used at post offices to recognize addresses on envelopes and sort them automatically.
Several researches have been carried out in this area previously. Most of the errors in recognition are due to the errors that occur during the segmentation phase [2,7]. The authors in [7] present the segmentation based on the feature extraction along with recognition using back propagation neural network. An accuracy of 90% was achieved. [3] has carried out the recognition using feedforward network followed by training using back-propagation neural network. It employs a 30 x 30 matrix of character as input to the neural network which then produces output on 49 neurons.

2. METHODOLOGY

There are basically three steps involved in the recognition of characters written in Devanagari script from an image of Devanagari text. The text is first segmented into lines, lines to words and words to characters. After the segmentation process, we then bring down the characters to a standard size so as to make the recognition size independent. This standard sized image matrix is then given to

the recognition module which employs Kohonen Neural Network to recognize the characters.

3. SEGMENTATION

Segmentation of the image is performed to separate the characters from the image [5]. Character Separation from the image of Devanagari Text involves the three steps which are described below. But before beginning with segmentation; let us first define two terms that will be used in the segmentation process.

Definition 1: Horizontal Projection, HP (k): For a binary image of size H*W, where H is the height and W is the width of image, horizontal projection can be defined as the number of black pixels in each horizontal row [1].

Definition 2: Vertical Projection, VP (k): For a binary image of size H*W, vertical projection can be defined as the number of black pixels in each vertical column [1].

3.1 Line Segmentation

The image of text may contain any number of lines. Thus, we would first need to separate the lines from the documents and then proceed further. This is what we refer to as line segmentation. To perform line segmentation, we take horizontal projection for every horizontal pixel row starting from the top of document. The lines are separated where we find a row with no black pixels [2]. That means, HP (k) = 0 where k is the row number where white space is found. This row acts as a separation between two lines (see Figure 1).

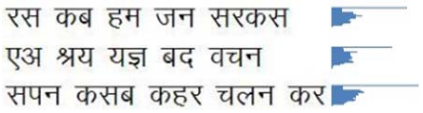


***Figure 1**: Line Segmentation*

3.2 Word Segmentation

After segmentation of lines from the text image, next task is to segment the words from the lines. This can be accomplished using the concept of vertical projection. If we take vertical projection for each line, then the words can be separated by looking for the column with zero black pixels [2]. That means, VP (k) = 0, where k is the column number where the white space is found. This k serves as the separating index for words (see Figure 2).

***Figure 2:** Word Segmentation*

3.3 Character Segmentation

To segment characters from the image, we need to use the words separated in the previous step and then find the position of header line *(Shirorekha)*. Once the header line is separated from the word, we can separate the characters individually. To locate the position of header line, we compute the horizontal projection of the word image box. The row that contains maximum black pixels corresponds to the position of the header line in the word [2] (See Figure 3). The characters can then be identified separately in the absence of header line.

***Figure 3**: Header Line Identification*

Now, we take the vertical projection of the word box below the header line. The columns that have no black pixels are treated as the boundary for separating characters from the word [2] (See Figure 4).

***Figure 4:** Character Segmentation*

After the segmentation process is complete, we obtain separate character boxes (See Figure 5) which can then be brought down to a standard size.

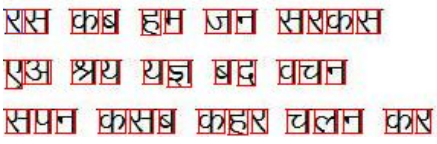


***Figure 5:** Final Segmentation Output*

4. DOWNSAMPLING

After we have the characters of the image of Devanagari text separated, we then bring the characters in a standard size. This is done so as to make the character recognition size independent. These can be brought to a standard size by defining the number of pixels of the character image box that are to be considered to calculate one pixel in down sampled image. This can also be referred to as windowing [3] (See Figure 6). This is done using the following algorithm [4]:

Step i) ratioX = (Width of character image box) / (Width of down sampled image)
Step ii) ratioY = (Height of character image box) / (Height of down sampled image)
Step iii) downsampledImage(x, y) = black, if there is a black pixel in the box starting from (x*ratioX, y*ratioY) to (x*ratioX+ratioX, y*ratioY+ratioY); white otherwise.

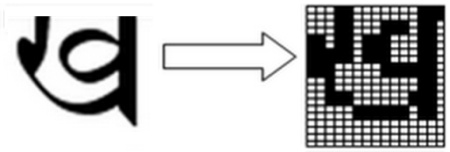


Figure 6: *Down sampling the character*

5. RECOGNITION

After the down sampled image matrix is generated, the next step is the recognition of the character. This recognition is done using Kohonen Neural Network or KNN [4]. The input neurons to the KNN are the elements of the down sampled image matrix. The output neurons are equal to the number of characters that can be recognized by the neural network. The input neurons are connected to output neurons through some weights. These weights define how well the network recognizes the input pattern. Using these weights and the input, we can calculate the output of the neurons by taking the dot product of the input neurons with the weights. The output neuron with the maximum value of output is chosen as the 'winner' and is identified as the output for that given set of input neurons. The corresponding character is then found by looking for the pattern in the training set that produces the same winning neuron.

For effective recognition, we have to train the network with a set of characters that the network can recognize. The training of the network will continue until the error of the KNN is below an acceptable level. Since KNN relies on unsupervised training, the error is not actually as it is defined with other networks. We have used a slightly different definition of error [4]. Let us define two terms that will be useful in calculation of errors.

Definition 1: NEURON_ON: This provides an optimal value for a neuron in its activated state. That is, it defines what the output value of a winning neuron is. Let us keep it at 0.9.
Definition 2: NEURON_OFF: This provides an optimal value for a neuron to be in its OFF state. That is, it defines what the output value of a neuron that hasn't won for a given input should be. Let us keep it at 0.1.
The error (E) can then be calculated by taking the mean of square of the difference of the output value of neuron (y_i) to either one of these two values. i.e., for winning neuron, it will be difference between the output of the neuron and NEURON_ON and for other neurons; it would be the difference between the output value of the neuron and NEURON_OFF [4].

$$E = \frac{\Sigma(y_i - NEURON_{ON})_{winning\,neuron} + \Sigma(y_i - NEURON_{OFF})_{other\,neurons}}{number\ of\ neurons}$$

The step by step algorithm for training of the network is described as follows [4]:
Step i) An input is presented to the software as training data. The training set is stored with the corresponding character that it represents.
Step ii) The training process begins by assigning the KNN Structure. The number of input neurons is equal to the size of down sampled image that is being given to the network. Let us consider three hundred input neurons.
Step iii) There are several training sets each representing a single character in Devanagari script. The number of training sets is the number of output neurons for the network.
Step iv) The initial weights between the input and output neurons are randomly generated real numbers between -1 to 1.
Step v) The next step is to normalize the input (x_i). First we find the vector length of input vector which is defined as the sum of squares of the elements of input vector. The normalizing factor is calculated by taking the reciprocal of square root of the vector length [4].

$$vector\ length = \Sigma x_i^2$$

$$normalizing\ factor = \frac{1}{\sqrt{vector\ length}}$$

The normalized input is then found by taking the product of input with the normalizing factor.
Step vi) The weights are then normalized using a similar algorithm by calculation of normalizing factor.
Step vii) The next step is the calculation of output of the neurons by taking the dot product of the input vector (x_j) with the weights (w_{ji}) of the output neuron [4].

$$Y_i = \sum_j x_j w_{ji}$$

The output is then normalized by multiplying with the normalizing factor calculated in step v.
Step viii) The output is then converted to bipolar by multiplying it by two and then subtracting one from it.
Step ix) The winning neuron is calculated by finding the output neuron having the highest output value among all the output neurons for a given training set.
Step x) Now, we modify the weights of winning neuron so that it reacts more strongly to the same input pattern the next time. For this, we define a learning rate (α) as 0.3 and decrease it by 1% after

each epoch. The weight adjustment is done using subtractive method[4].

$$w_t = w_t + \alpha \times (x_t - w_t)$$

Step xi) If there exists a neuron that fails even to learn, then it must be forced to win for at least one input pattern [4]. This is because for every input pattern, we have one output neuron to the network. For this, we go through the entire training set and find which training set pattern causes the least activation.

Step xii) The training set identified in the previous step is then chosen as the training set which is least well represented by the current set of winning neurons.

Step xiii) The values of output neurons for this training set is now calculated and the neuron with the maximum output value among the neurons that haven't yet won is selected as the neuron which best represents the input neuron and whose weight we will modify to better represent the input pattern.

Step xiv) The weights of that neuron are then modified so that it better recognizes the input pattern.

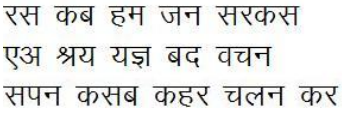
Step xv) The training process stops when the error is below a desired level.

6. RESULTS

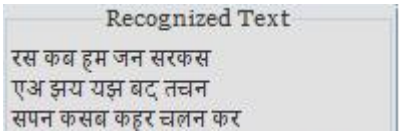
We have trained the software using standard printed characters as well as for handwriting. For the printed text input, the accuracy is 90.26% (See Table 1).

Table 1: *Accuracy for image of printed Devanagari text*

ID	Total Characters	Correct recognition	Font (Size)	Accuracy
1	*275*	*254*	*Kruti (20)*	*91.49%*
2	*344*	*306*	*Kruti (24)*	*89.02%*
3	*235*	*212*	*Unicode(17)*	*90.29%*
Average Accuracy				*90.26%*



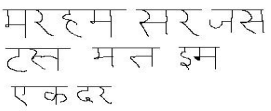
a) *Input Image of Printed Devanagari Text*



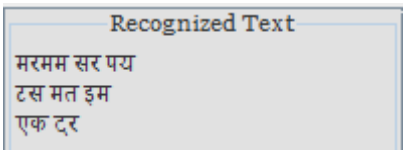
b) *Corresponding recognized text*

Figure 7: *Results on printed Devanagari text*

We can provide the image of printed Devanagari Text as input to the software for recognition (See Figure 7).



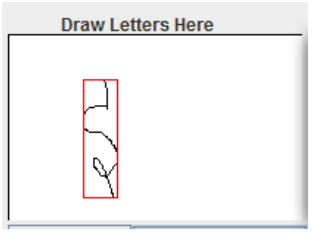
a) *Input Image of Handwritten Devanagari Text and*



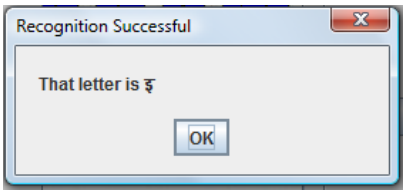
b) *Corresponding Recognized Text*

Figure 8: *Results on handwritten Devanagari text*

We can also provide the image of handwritten Devanagari Text as input (See Figure 8). For the above text the accuracy is 83.33%. It is our endeavour to make the segmentation and pre-processing algorithms discussed in this paper more robust to cater various types of complexities in handwritten text.



a) *Sketched Devanagari Character*



b) *Corresponding recognized character*

Figure 9: *Results on sketched Devanagari character*

We can also sketch Devanagari characters in the drawing area for recognition (See Figure 9).

7.CONCLUSION

The method for recognition of Devanagari characters presented in the paper is able to recognize most of the given text and also recognize the sketched Devanagari character presented to it. Success also depends on the training of the neural network. Higher the training, higher would be the accuracy.

8. ACKNOWLEDGEMENTS

We would like to thank our Director Dr. M.D. Tiwari for providing excellent computational facilities and

stimulating work environment for carrying out the research work.

9. REFERENCES

Liang, S., Sridhar, M. and Ahmad, M. (1994) Segmentation of Touching Characters in Printed Document Recognition, *Pattern Recognition*, 27, pp. 825-840

Sinha, R.M.K. and Bansal, V. (1995) On Devanagari Document Processing, *IEEE International Conference on Systems, Man and Cybernetics, Canada*, Vol 2, pp. 1621-1626

Rajput, K.Y. and Mishra, S. (2008) Recognition and Editing of Devanagari Handwriting using Neural Network, *IEEE Colloquium and International Conference, Mumbai*, Vol 1, pp. 66-70

Heaton, J. (2008) *Introduction to Neural Networks for Java*, 2nd ed., Heaton Research

Gonzalez, R.C. and Woods, R.E. (2002) *Digital Image Processing*, 2nd ed., Pearson Education (2009) Languages of India, http://en.wikipedia.org/wiki/Languages_of_India (Accessed: Dec. 8, 2009)

Yadav, D., Sharma, A.K. and Gupta, J.P. (2007) Optical character recognition for printed Hindi text in Devanagari using soft-computing technique, *IASTED International Multi-Conference: Artificial Intelligence and Applications, Innsbruck, Austria,* pp. 102-107