Optical Character Recognition for Handwritten Hindi

Aditi Goyal, Kartikay Khandelwal, Piyush Keshri
Stanford University

Abstract

Optical Character Recognition (OCR) is the electronic conversion of scanned images of hand written text into machine encoded text. In this project various image pre-processing, features extraction and classification algorithms have been explored and compared, to design high performance OCR software for ***Indian Language Hindi*** *based on Devanagari script. The best performance obtained with handwritten letters is* ***98.8%*** *using One-against-All SVM technique and extracting features using HOG (Histogram of Gradient) technique.*

1. Introduction

1.1 Motivation

OCR finds wide applications as a telecommunication aid for the deaf, postal address reading, direct processing of documents, foreign language recognition etc. This problem has been explored in depth for the Latin script. However, there are not many reliable OCR software available for the Indian language Hindi (Devanagari), the third most spoken language in the world. [1] provides a good starting point for the problem and presents a good overview. The objective in this project is to design high performance OCR software for Devanagari script that can help in exploring future applications such as navigation, for ex. traffic sign recognition in foreign lands etc.

1.2 Framework Description

1.2.1 Hindi Language Fundamentals

The Hindi Language consists of 12 vowels and 34 consonants. The presence of pre and post symbols added to demarcate between consonants and vowels introduces another level of complexity as compared to Latin script recognition. As a result, the complexity of deciphering letters out of text in Devanagari script increases dramatically because of presence of various derived letters from the basic vowels and consonants. In this project emphasis has been laid on recognizing the individual base consonants and vowels which can be later extended to recognize complex derived letters & words.

1.2.2 Dataset Generation

Because of the limited scope of work being done in this realm, standard hand written dataset for Hindi is not readily available. Hence, the entire dataset has been generated by taking hand written samples from 20 different users.

1.3 Approach

The approach followed during the project was to formulate a large systematic standard dataset, extract important features from the scanned text images and to implement high performance off-the-shelf classification algorithm. The following sections discuss about the methodology implemented throughout the project to improve the performance of the system.

2. Methodology

2.1 Training Set Generation

The handwritten data set was manually generated for each of the 46 fundamental characters. To standardize image preprocessing steps, a standard template was created consisting of 88 blocks (300 x 300 pixels each) on a 8.5"x11" sized sheet, where each block contained exactly one character. Samples from 20 different users were obtained to account for varied human calligraphy style and fonts sizes. While generating samples, angular skewness was avoided. All characters were written with a standard black pen. These sheets were then scanned to generate the character images. A complete dataset of ~150-200 samples for each of the 46 characters was acquired, resulting in a dataset of ~**8000 characters**.

2.2 Image Preprocessing

The scanned image was first converted from RGB scale to gray-scale. It was then splitted into individual character blocks using MATLAB script to obtain raw individual character samples. The following preprocessing and noise removal techniques were used on raw samples to obtain a clean dataset.

2.2.1 Median Filtering. Scanning process introduces irregularities such as 'speckle noise' and 'salt and pepper noise' in the output image. Median Filtering was employed, to remove such effects, where each pixel was replaced by the median of the neighboring pixels.

2.2.2 Background Removal. To model the background noise due to scanning, a white page was scanned with the same scanner and this image was subtracted from each of the character images, hence eliminating background and any residual background noise; highlighting only the character sample.

2.2.3 Thresholding. To remove any residual irregularities and to increase image contrast, all pixel values above 200 were scaled upto 255. Also, all pixels lying at the boundary within a 50 pixel wide strip were scaled upto 255 to ensure a clean boundary.

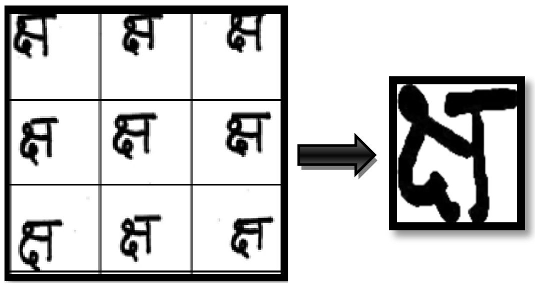

Fig1. Image after PreProcessing.

2.2.4 Sparsity Removal. It was observed that the image matrix was sparse and the character size within the image was much smaller than the complete image. Hence, a MATLAB script was written to create a tight bounding box around the character and to extract pixels into a 128px*128px matrix thus increasing the density of useful character information.

After creating individual character images by the above process, any abnormalities in the data set were also removed manually.

2.3 Feature Selection/Extraction

The following feature extraction methods were employed and tested on a training set to determine the most optimal set of features for our specific handwriting recognition problem.

2.3.1 Raw Pixel Data

The most basic feature set for any image is its pixel intensities, and thus this was the first set of features that were employed. While these set of features are easy to design, they lead to a very high dimensional feature vector (16,384 dimensional for a 128px*128px image). Reducing the image size to extract reasonably sized feature vectors blurs the image and leads to loss of information. However, a large feature vector size results in massive learning time of high complexity classification algorithms such as SVM and also results in over-fitting (high-variance) issues.

2.3.2 Histogram of Oriented Gradients (HOG)

To overcome the problems associated with the features generated from raw pixels, Histogram of Oriented Gradients (HOG) features [2] were used. This feature set is independent of the image size and captures localized information about intensity gradients. The HOG window size and the number of bins in the histogram can be varied to analyze the performance of classification with respect to feature size. Thus, this provides a flexible set of representative features and helps to deal with both high bias and high variance issues.

Design Parameters for HOG features
Number of HOG windows per bound box:
Along Horizontal = 3;
Along Vertical = 3
Number of Bins in Histogram = 9
Filter Kernel:
Along Horizontal = [-1 0 1]
Along Vertical = [1 0 -1]'

2.3.3 Sparse Autoencoder

The third method being analyzed in the design process for feature extraction was Sparse Autoencoder, a multi layered neural network. As in [3], the basic Back-propagation algorithm was used to determine the weight matrix.

After the weight matrix (30x64) was obtained, each image was divided into 16 matrices (patch size 8x8) and an estimate of the feature vector (length = 30) for each patch was calculated. For every image, the vector with maximum norm was chosen as the feature vector.

Design Parameters for the Sparse Autoencoder
Activation Function: Hyperbolic Tangent
# of inputs = 64
#of Layers = 3
# of Hidden Units = 30
Weight Decay Parameter (lambda) = 0.002
Learning Parameters: **alpha** = 0.003 **beta** = 5
Sparsity Parameter: rho = -0.996
Number of Iterations = 5 million

2.3.4 Analysis of Feature Extraction Techniques

Firstly, the above features were tested on a reduced Training Set (4 character labels) using Naïve Bayes as shown in Table1.

Observations

- Since the Raw Pixel data was very high dimensional, the run time for these features was very high. As expected, the larger image matrix (128px*128px) performed better in terms of accuracy. Plotting the smaller image showed that a large amount of the information was lost as a result of blurring effects.
- The performance of HOG features was found to be much better than the Sparse Autoencoder in terms of test error. This can be explained by the fact that the HOG features best captured the distribution of intensity gradients and edge directions. It was also observed that the Sparse Autoencoder had higher run-time for both training the neural network and extracting the features from individual images; making its realization difficult for large feature vector dimension.
- Increasing the size of the HOG feature vector led to higher test errors primarily because of over-fitting of parameters.
- Higher test errors were obtained with raw pixel features as compared to HOG features showing that most of the character information is contained in the gradient of intensity and not the absolute intensity values.

Owing to better accuracy and low run time, HOG features were used for subsequent analysis and Classifier Formulation.

Feature Extraction	**Test Error**
Raw Pixel (Image Size 128px*128px)	18.56%
Raw Pixel (Image Size 16px*16px)	45%
Sparse Autoencoder	14.30%
HOG (Feature Vector Dim. = 81)	5.40%
HOG (Feature Vector Dim. = 108)	7.80%

Table1. Test Error for Different Features.

2.4 Classifier

After selecting the feature extraction technique, while choosing the classifier algorithm for OCR the following three classifiers were analyzed.

2.4.1 Naïve Bayes

For a quick and dirty implementation, the Naive Bayes Algorithm was used. Since the pixel values and hence, successive features are dependent on the character label, the Naïve Bayes Assumption (features being conditionally independent of output labels) may not hold. This explains the low accuracy that was obtained with this classifier as shown in Fig 2.

2.4.2 Support Vector Machines

To use SVM in the multi-class case, the 'SVM and Kernel Methods' MATLAB toolbox was used. In particular, the '**One against All'** approach was used. The errors for Gaussian and Polynomial Kernels were evaluated. The variation in error with modification in the cost parameter 'c' were also observed. Specifically, decreasing 'c' from 1000 to 500 and then to 50 and 5 led to an increase in the test error as well as an increase in the running time of the algorithm. Increasing 'c' from 1000 to 10000 showed a similar increase in test error.

<u>Final parameters used:</u>
Cost Parameter,c = 1000
Lambda, $\lambda = 10^{-7}$

2.4.3 Adaboost

To use Adaboost in the multiclass model, the 'Gentle Adaboost for Multiclass Classification' MATLAB toolbox was used. In particular, we used the Decision Tree as the underlying base learner. Using the 'Perceptron' as the base learner led to very high run times of the algorithm and hence this weak learner was not used on the entire dataset.

<u>Final Parameters used:</u>
Lambda, $\lambda = 0.02$
Epsilon, $\varepsilon = 0.1$

3. Test Results

For evaluating the final errors for each classifier, we used the **LeaveMOut Cross Validation** with M (size of test set) fixed at 400 samples. This was done for 10 iterations with the test set being randomly chosen in each iteration. The following curve shows the plot of test and training errors for SVM (Gaussian kernel) as a function of training set size.

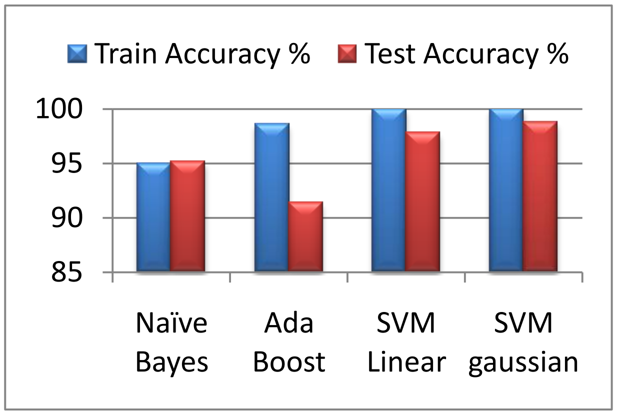

Fig 2. Test and Training Error over whole dataset.

3.1 Observations and Conclusions

- The Naïve Bayes classifier gives high errors probably because the Naïve Bayes assumption of x_i^j being conditionally independent of y_i does not hold here.
- The test error for Adaboost (fig3.) was seen to be surprisingly high for an iterative algorithm trying to minimize error in every iteration. This can be attributed to the use of a binary weak learner such as Decision Trees, which seem to break down with the increase in the number of classes.
- As seen in fig3., SVM shows higher test error for smaller training data but **asymptotically gives better results** than either Naive Bayes or AdaBoost.
- *The One-againstAll SVM approach with Gaussian kernel (cost parameter = 1000) was found to have the highest test accuracy of 98.8%.* This method also gave 100% accuracy for the training set (fig. 4). One possible explanation for this seems to be that the HOG features (with regularization) give linearly separable data points that can be correctly separated by a maximum margin hyperplane.
- The SVM Linear kernel gave about 1% more error than the Gaussian kernel probably because the dataset was linearly separable

even in a finite feature dimensional space and hence using a Gaussian kernel did not yield significant improvements.

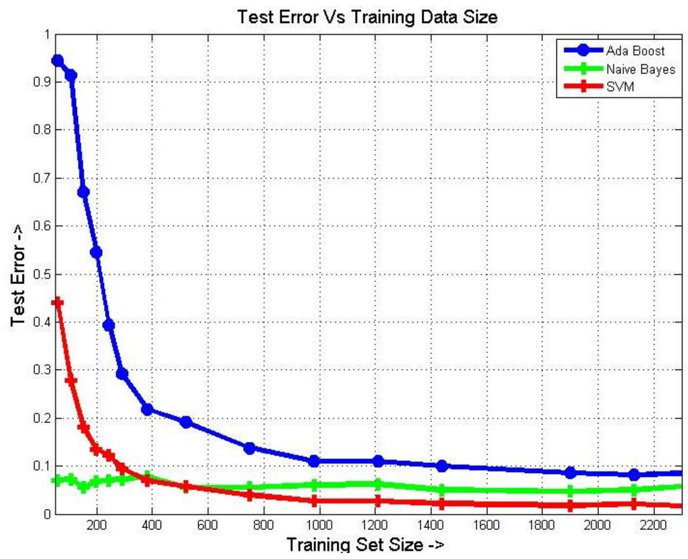

Fig3. Test Error Vs Training Data Size

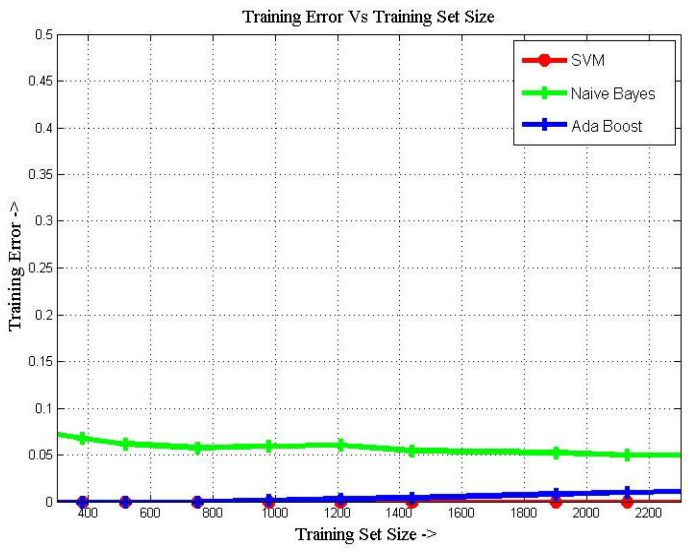

Fig4. Training Error Vs Training Data Size

- Some of the reasons of getting high value of accuracies with SVM could be:
  i. Linearly separable data set - As shown in [5], SVM achieves zero training error for linearly separable dataset and very high accuracies for the test set. This basically implies that the intra-class features were kind of similar while there was significant difference between inter-class feature vectors.
  ii. Clean data set – It was ensured that the dataset contained no rotated / distorted images.
  iii. Simplistic character set – The project recognizes basic consonants and vowels instead of complete words.
- The confusion matrix showed that some particular combinations of characters were being confused consistently by all the three classifiers. These were also verified to be visually similar to the human eye.

5. Acknowledgements

We thank Prof. Andrew Ng, Andrew Maas and Quoc Le for their guidance and support throughout the project. We would also like to thank all those who contributed towards the generation of our data set.

6. References

[1] C.V. Jawahar, R. Kiran, "A Bilingual OCR for Hindi-Telugu Documents and its Applications," ICDAR, Aug.'03,Vol.2.

[2] N. Dalal, B. Triggs, "Histogram of oriented gradients for human detection," Conference on Computer Vision and Pattern Recognition, 2005, Vol. 1, pp. 886-893.

[3]] A. Ng, "CS294 Notes - Sparse Autoencoder," Winter'09, Stanford University.

[4] M.S. Jelodar, M.J. Fadaeieslam, N. Mozayani, M. Fazeli, "A Persian OCR System using Morphological Operators," World Academy of Science, Engineering and Technology, 2005, pp. 241-244

[5] A. Ng, "CS229 Notes - Support Vector Machines," Fall'10, Stanford University.