Optical Character Recognition for Devanagari Script (Final Report)

Anand
2017218
IIIT-Delhi, India
anand17218@iiitd.ac.in

Akash Singh
2017013
IIIT-Delhi, India
akash17013@iiitd.ac.in

Atul Anand
2017284
IIIT-Delhi, India
atul17284@iiitd.ac.in

Abstract

Optical Character Recognition systems are common these days. However, most of such system supports english language only. Devanagari script used by around 120 Indo-Aryan languages is popular among huge population. In this paper, we presented a deep learning architecture based OCR system for Devanagari Script. Publically available dataset for training the model. Initially we started with basic models such logistic regression and SVM. To increase the accuracy we moved to deep learning architecture. Convolution Neural Network being one of the most popular resulted as the best performing model. Some optimizations (including dropouts, batch normalization) and hyper-parameter tuning help in increasing the accuracy by around 4%. The best accuracy obtained is 99.03% with combination of CNN and SVM.

1. Introduction

Devanagari Script was developed in ancient India from 1st to 4th century CE [1]. It is used by around 120 Indo-Aryan Languages and widely popular in Indian subcontinent. Most of the optical character recognition are limited to english langauge only which is generally used in west. Considering the benifits of such system for asian (Indian subcontinent) context, we aim to develop a Machine Learning model which identifies any handwritten character as one of the 46 characters available in Devanagari Script. It would work as an OCR system as well as a benchmark for further research and work in this field. It would also draw the attention of researchers and software system practitioners to work towards in this domain.

Source	Model	Accuracy
Acharya et al. [2]	CNN	98.47%
Kabra [3]	CAE+SVM	96%

Table 1. Previous works

Unlike Latin characters used in English, all Devanagari characters of a word are written in a hanging baseline without inter-character separation. So, developing OCR for such script can be challenging. This OCR system can help editing and economic sharing of existing printed texts of this script, which is otherwise not possible with their scanned counterparts.

2. Related Work

Machine Learning including deep learning architectures has been explored for character recognition and researchers have used various algorithms and feature extraction methods to improve the performance of the model.

The table 1 shows the state of art work on using machine learning models for Devanagari character recognition dataset. Acharya et al.[2] , in his paper, originally made the dataset publically available. They presented two different deep learning models (few parameter different) based on convolution neural network with relu activation function. Different experimental alterations and setting optimal dropout rate resulted in the accuracy of 98.47%. Similarly, Kabra [3] applied contractive auto encoder along with SVM to achive the accuracy of 96%.

Name	Number
Total samples	92000
Training samples	78200
Test samples	13800
Features	1024
Classes	46

Table 2. Dataset information

3. Methodology

Figure 1. Sample image containing 32x32 gray scale pixels including 2 pixels of padding from each side

The total dataset [4] has 92000 samples with 1024 (32x32) features. The data is divided into training and testing in the ratio 85:15 having 78200 and 13800 samples respectively. We used randomization for better validation and uniformity of data samples used to train data. The table 2 gives a summary about the information of the data. Figure 1 represents a sample image from the dataset representing character 'kaa'.

3.1. Feature Extraction

Initially, we removed the 2 pixels of padding from each side of 32x32 images and hence reduced size of image to 28x28. In also reduced the number of features from 1024 to 784.

For non deep learning models, correlation between feature and result was enquired for all features. It was found that around 23% of data had almost no correlation with results. Another 6% features were having low correlation (less than 0.01). This was mostly because already there was a padding of 2 pixels from each side(240 pixels used in padding) and anyhow, letter is mostly placed in centre of image, hence only *central features* contribute high. The removal of these features helped in decreasing features further from 784 to 722.

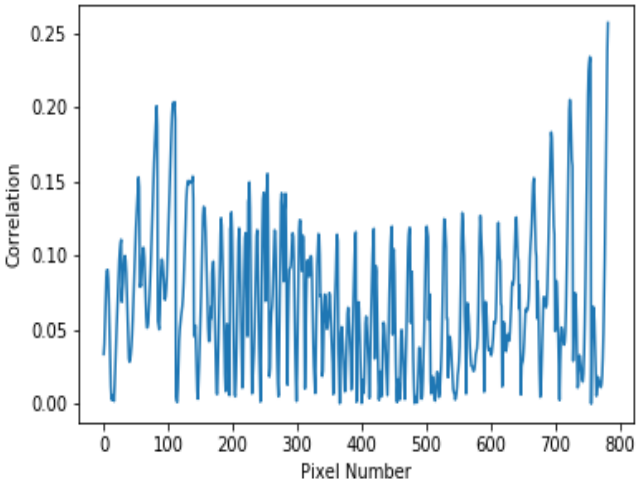

Figure 2. Correlation of features with output variable (after dropping padding)

However, for deep learning models, 784 features were used since hidden neurons essentially work as feature detectors and reduce the feature on their own. Batch normalization layers and dropouts used also helped in further finding the best features.

3.2. Evaluation Metrics

For our dataset, accuracy is a sufficient measure of the performance of the model as the dataset is perfectly balanced. Each class has equal number of samples, hence we can rely on accuracy as an evaluation metric. Apart from accuracy, losses were also taken into consideration wherever suited. Since the number of classes are very high, we are not going to use other evaluation metrics like confusion matrix and ROC-AUC curve as it will result in high computational complexity and would be harder to visualize in our case.

4. Results

The main objective was to get the best possible accuracy. For that we begin with simple machine learning models and them moved step by step to advanced models.

We started off with linear models such as logistic

Model	Accuracy
Logistic Regression	65.9%
Random Forest	83.1%
SVM with Linear Kernel	74.44%
SVM with RBF Kernel	82.44%
SVM with degree 2 Polynomial Kernel	82.52%
KNN	83%

Table 3. Machine Learning models and accuracies

regression which gave average results. For logistic regression, we used different optimizers on a random shuffled subset of data and used the best model (optimizer and decision function) for training on the whole data. We ruled out Logistic as this is not be optimal for this data set.Then, decision trees were used which again gave average results on Test Set due to overfitting on training set. Later, Random Forests were used which use Ensemble Learning, hence giving better results. AUC scores of random forest and gradient boosted trees tell that both the trained models are equally good but the false positive rate is lower for gradient boosted trees and accuracy is higher. Later, we also employed K Nearest Neighbor (KNN) and Support Vector Machine (SVM).

Table 3. gives a summary of the results obtained with these classifiers. Figure 3 and 4 represents the change in accuracies for some of these models (random forest and KNN) with varying the parameters involved. For KNN, it turns out that increasing the number of layers resulted in decrease in accuracy. Setting number of neighbours to 1 gave the best accuracy for KNN. In the case of random forests, increases number of trees has not much impact on accuracy, however increasing depth resulted in sharp increase the accuracies. The best result was obtained for trees 175 or greater and depth 16 or greater.

Later on to further improve the accuracies, deep learning (neural network) based algorithms were used. We used CNN, CNN coupled with SVM. Apart from CNN, we also used contractive autoencoder (CNN) with SVM. In CNN, we first ran using different optimizers for 15 epochs. Out of all the 7 (adam, adamax, adagrad, adadelta, nadam, rmsprop and sgd) optimizers, adadelta turns out to be best performing optimizer due to its adative hyperparameter tuning. Cat-

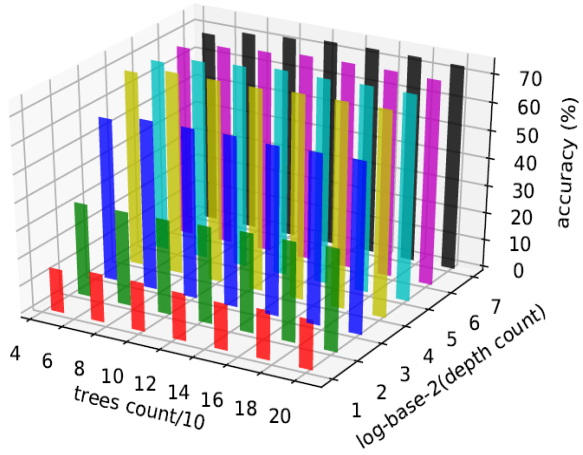

Figure 3. Random Forest Classifier with different tree counts and depth

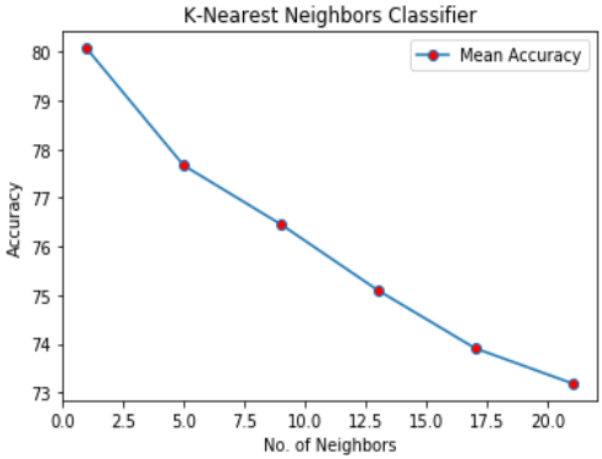

Figure 4. K-Nearest Neighbour classifier with different K values

egorical loss entropy function is used for all the models since it works best with multi-class classification and three (conv + relu + max pooling) were used. The second step was finding the best set of parameters. On increasing epoch to 50 and patience to 6 and decreasing layer to two (conv + relu + max pooling), the accuracy of 0.97500 was obtained. Replacing max poolig with average pooling further pushed it to 0.97572. Henceforth, this model was used. Increasing kernel size, strides, pool size resulted in slower training and decreased accuracy, therefore, they are forbidden. Next, drop outs were introduced in both the set of layers. With the dropout rate of 0.5, accuracy achieved was 0.98609. Finally, dropouts were replaced with

Model	Accuracy
CNN	98.83%
CNN + SVM	99.06%
CAE	95.09%
CAE + SVM	96.22%

Table 4. Deep learning models and accuracies

batch normalization which increased the accuracy to 0.98826. It was the best accuracy achieved from CNN.

In the next model which was output from penultimate layer was passed to SVM as training data. Using SVM with rbf kernel and one versus rest decision function resulted in 0.99058 accuracy. SVMs are know to perform better when feature extracted inputs are passed to them.

Lastly, contractive auto encoder gave and accuracy of 0.950942 and when it is coupled with svm it increased to 0.96215 which is similar to the results obtained for Kabra. Table 4 shows the accuraccies for these models. Figure 6 and 8 depict Accuracy of CNN model by varying optimizers and dropout rates respectively.

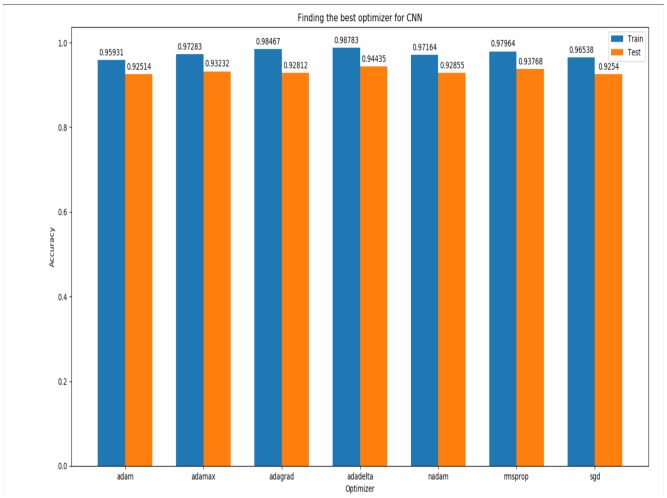

Figure 5. CNN accuracy with different optimizers

5. Conclusion

In this paper, we used several machine learning models to train and correctly classify the dataset of Devanagri script characters. The training was a challenge since in Devanagari, characters of a word are written in hanging baseline without inter-character space separation unlike the English letter. Non deep learning models turned out to be giving lesser accuracy than deep learning models. Deep learning models especially CNN turns out to be best performing model. Batch normalization further increased the accuracy on test set. Using average pooling and batch normalization could be the reason for the same since previously the original author used max pooling and dropout rate respectively. The final accuracy obtained is 99.06% using CNN extracted features passed to SVM (rbf kernel) which is greater than the accuracies of the work we mentioned earlier. SVM performed much better when extracted features were passed to it.

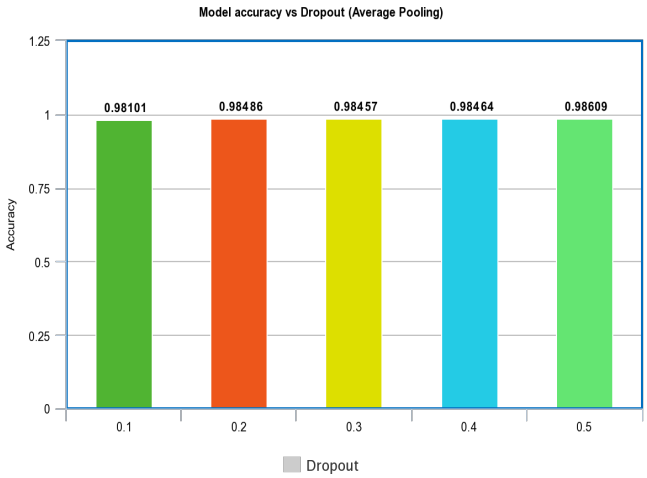

Figure 6. CNN accuracy with different dropout rates

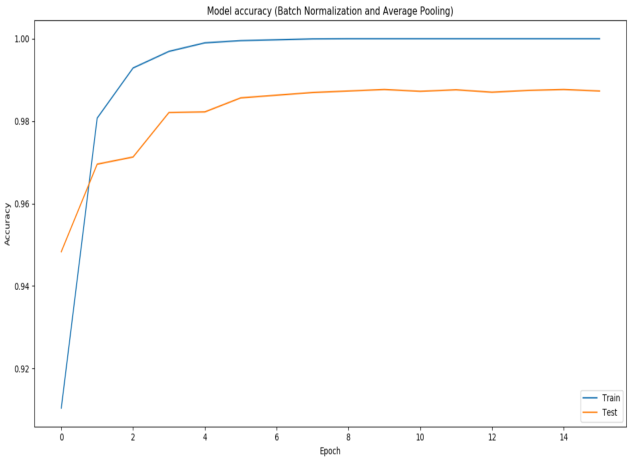

Figure 7. CNN accuracy - Batch Normalization and Average Pooling

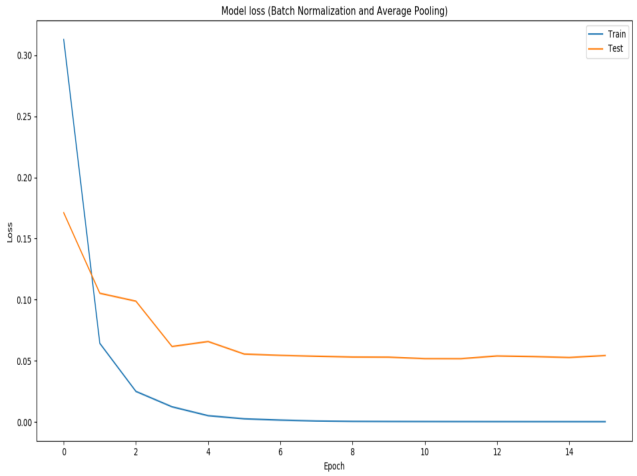

Figure 8. CNN loss - Batch Normalization and Average Pooling

References

[1] "Devanagari," Wikipedia. 28-Nov-2019.

[2] S. Acharya, A. K. Pant, and P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition," in 2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), Kathmandu, Nepal, 2015, pp. 1–6.

[3] R. R. Kabra, "Contractive autoencoder and SVM for recognition of handwritten Devanagari numerals," in 2017 1st International Conference on Intelligent Systems and Information Management (ICISIM), 2017, pp. 26–29.

[4] "UCI Machine Learning Repository: Devanagari Handwritten Character Dataset Data Set." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Devanagari+Handwritten+Character+Dataset/.