Optical Character Recognition of Jutakshars within Devanagari Script

Sheallika Singh : 12665
Shreesh Ladha : 12679

May 3, 2016

ANNEXURE-II

DECLARATION

I/We hereby declare that the work presented in the project report entitled ``……" contains my own ideas in my own words. At places, where ideas and words are borrowed from other sources, proper references, as applicable, have been cited. To the best of our knowledge this work does not emanate or resemble to other work created by person(s) other than mentioned herein.

The work was created on this ….. day of …….. 20…..

Name and Signature Name and Signature

Date:

Abstract

Optical Character Recognition (OCR) is a technique of converting document images, scanned documents and PDF files into editable text. It is also widely used in document handling applications, reading serial numbers in automotive or electronic applications, aiding visually impaired people to read and many more. A lot of work has been done on OCR for roman script, but similar softwares for hindi languages with high precisions are yet to be developed. It is difficult to create robust systems for Devanagari because of many of its complicated features. One such feature which adds to the complications is a class of characters called Jutakshars (or Samyutakshars). In this project we have tried to detect and identify these jutakshars within words and have obtained encouraging results.

Contents

Chapter 1

Introduction

Optical character recognition widely known as OCR, has allowed us to take great steps forward towards tackling one of the hardest challenges faced by computer users in last few decades. OCR is a tool for converting printed text into editable and human readable text. Quite extensive work has been done in reference to OCR systems for Roman script and there are a lot of softwares which can recognise characters and words with high accuracies. However, there is a lack of such systems for Devanagari script (which forms the basis of several Indian languages like Hindi, Sanskrit, Marathi) because of their highly complicated structure and composition.

An example of such a composition is a form of character conjunction referred to as Jutakshar. A Jutakshar is basically a concatenation of two characters - one half and one full character. Some examples have been shown in Figure 1.2. Words containing Jutakshar are quite popular and can be found many times within a document. Still, most existing models of Hindi OCR do not take special care to handle such characters, which in turn reduces their precision. This happens because such characters get recognized as a single character which badly affects the overall word/char prediction rate. In our project, we have tried to build upon existing frameworks to better handle these special strings. Note that various forms of Jutakshars exist in Hindi Language but well be concentrating on only one class where the half character precedes the full character in horizontal direction (example shown in image). A basic working of an OCR model is described below. We will not be talking in detail about its functioning and it is assumed that the reader is aware of it while going through our report. More details of it can be found in [1]

The Hindi OCR model that we are working with, starts off with some basic

preprocessing of the image. The preprocessed image is used to extract lines by creating horizontal histograms and locating gaps. Words are extracted from within those lines using vertical histograms and finding gaps. The word is broken up into three regions : the part above the shirorekha , the part in the middle and the part at the bottom. This is better explained in the image below. Now, a vertical histogram is used again within each of these regions to segment the different characters. These segmented characters are fed to three pretrained classifiers, each for the three different regions, and the predictions are then combined according to the rules of the Devanagari script to finally obtain a word.

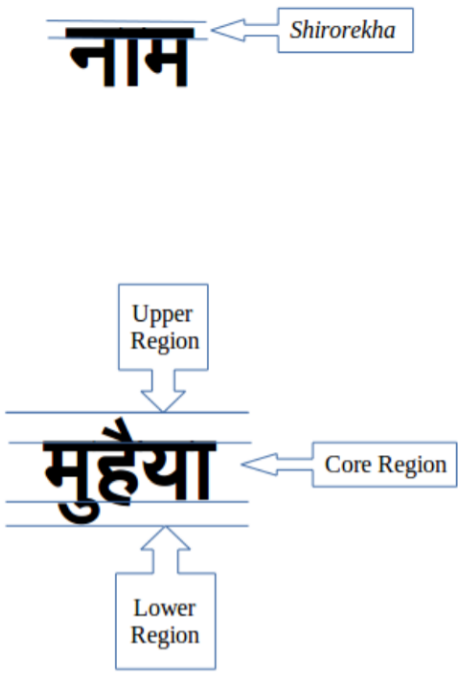

Figure 1.1: Shirorekha and 3 regions of a word: Upper, Core and Lower[1]

क्क च्च च्ज ज्य श्च श्व म्न प्र फ्ल ब्ब

Figure 1.2: Different types of Jutakshars (We have worked on the ones similar to the first four)

कश्मीर के कुलगाम जिले में भारतीय सेना ने एक मुठभेड़ में दो आतंकवादियों को मार गिराया। रविवार की शाम से ही कुलगाम के रेडवानी में मुठभेड़ चल रही थी और आतंकी एक घर में छिप कर फायरिंग कर रहे थे। इस बीच, मुठभेड़ में एक आम नागरिक की भी मौत की सूचना है। इसके बाद स्थानीय लोगों ने विरोध प्रदर्शन शुरू कर दिया है। मुठभेड़ में मारे गए दोनों आतंकियों के संबंध आतंकी संगठन लश्कर से बताए जा रहे हैं। इस दौरान सेना के दो जवान भी घायल हुए हैं।

Figure 1.3: Sample Devanagari Text[1]

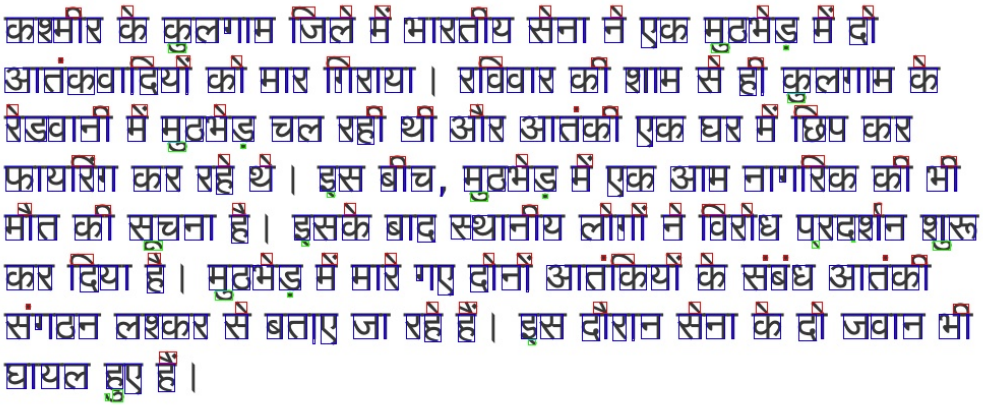

Figure 1.4: Example of Segmentation[1]

Chapter 2

Approach

Our plan of action was to first predict whether a word contained a Jutakshar or not and then proceed to correctly recognize the Jutakshar.

2.1 Dataset Formation

In this regard, we first created a dataset of images of Jutakshars normalized to a size of 20x20. For this step we created an image containing all the jutakshars and used the same algorithm that was used to extract characters from an image to extract these individual jutakshars. In order to make our model more robust, noise was added to the images before training the model. We added images with 2 types of noises:

- Gaussian noise - noise having probability density function equal to that of gaussian distribution (normal distribution) and added smartly based on the density of pixels.
- Adaptive mean thresholding - Thresholding based on the mean of the neighboring values.

A set of 60 such characters was used(not exhaustive) and copies of it were generated in eight different fonts(Chandas, Devanagari, Gargi, Guruma, Kohinoor, Lohit, Osho, Sarai) of the Devanagari family. A dataset for characters was already present from a preexisting model.

2.2 Jutakshar Detection

Now, we trained classifiers to predict whether a given character was a Jutakshar or not. Simple features such as pixel values are unable to capture

the different characteristics of images. Hence, we started off by using Hog or Histogram of Gradient features. Intuitively Hog tries to capture the shape of structures in the region by capturing information about gradients. It does so by dividing the image into small blocks of cells. Each pixel in the cell then votes for a gradient orientation bin with a vote proportional to the gradient magnitude at that pixel. The first model(a)(SVM) was trained using these hog features.

The second classifier(b)(logistic) was trained using features extracted from the penultimate layer of a pretrained convolutional neural network, bvlc_reference_caffenet, trained on ILSVRC 2012(ImageNet 2012) by Krizhevsky et al.. The features were extracted using Caffe, a deep learning framework made with expression, speed, and modularity in mind which provides models for different tasks with all kinds of architectures and data[2]. The images that were used to train bvlc_reference_caffenet were completely different from the images that we are dealing with. However, it has been observed that such deep models, through their subsequent layers, are able to extract relevant features from images which work well for many computer vision tasks and hence was the reason for our choice.

Finally, the third classifier(c) was a very simple model, using the width of the characters to classify it correctly into the two categories. We used SVM for (a) as it was performing better overall than other classifiers. While for (b), we chose a logistic classifier since we required the probabilities for fusing the above models. But, the final model used a combination of only the above two classifiers and predicted by a simple majority voting. Incase of a conflict, if (b) rejected then we rejected. Otherwise if (b) accepted with a high probability, we accepted. The model trained on hog features did not give high accuracy, as is visible in the image(Figure 2.4), and hence was not used in the final model.

2.3 Jutakshar Identification

The previous Jutakshar Detection model detected words containing jutakshars. Now we moved onto creating a model to predict these words and in turn the Jutakshars.

For building this model, the words were (as usual) segmented into characters. Owing to the structure of jutakshars and the algorithm used in segmentation, the jutakshars were being segmented as one single character. To handle

this issue, we modified the algorithm used in the segmentation of words into characters so that even the jutakshar would get separated into the half and the full character forming the respective jutakshar. For such characters, we used similar vertical histograms to locate gaps, but we restricted our search in the middle part of the character. This was achieved by creating a histogram in a window length of w/2 (where w = width of the character box) whose starting and ending points lay w/4 and 3*w/4 pixels away from the beginning position. The position where the minima was found was used to split the the Jutakshar into the half and the full character forming it.

Now, this segmented word is passed onto the character classifiers to predict the respective characters (also trained on the penultimate features of the deep model). Note that even the half character is predicted as a full character using this model. A dictionary is relied upon to resolve this issue. In that regard, we use a Hindi dictionary provided by Aspell, which gives a list of suggested correct words for an incorrect word. It doesnt just compare the words but also uses phonetic comparisons with other words and for this reason is able to give good results. Coming back to our procedure, our predicted word is now fed to the dictionary and a heuristic is defined to find the correct prediction from the suggested list.

A first simple filter for this list is to only look for words which contain 'Halant'. Then, we assign a score to all such words and the one with the minimum score among them is chosen as the final prediction. Score assigned to each word is a weighted average of :

- length of the longest common substring between the predicted word with the jutakshars removed and the suggested word in the list,
- number of substrings that match in the predicted word with the jutakshars removed and the suggested word,
- number of different characters between the predicted and the suggested word,
- and the levenshtein distance (computes the minimum number of steps required for converting one string into another by insertions, deletions or substitutions of single characters) between the predicted and the suggested word

The exact expression was :
Score $= 2 * (1) + 3 * (2) - abs((3) - (4))$

Instead of using a simple regular expression, we used this heuristic which we found to be more flexible. Also, the prediction of the characters apart from the ones constituting the jutakshar are believed to be fairly accurate. Therefore, only that part was used while using the heuristic to choose the final prediction from the list provided by the dictionary. The weights assigned to the above features may not be the most optimal, however, worked fine on our dataset. A better heuristic and weightage could be designed.

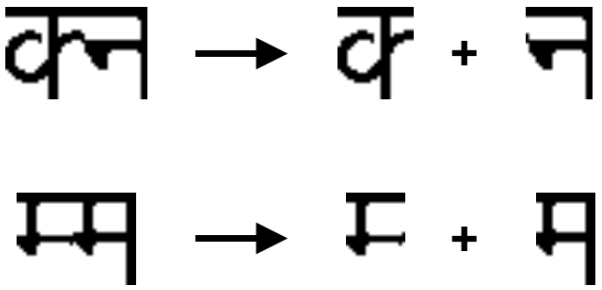

Figure 2.1: An example of how splitting works

Owing to the great performance of the jutakshar identification models(using features of pre-trained deep models) we tried experimenting with our own deepnet models. For this purpose, we trained end-to-end models using our data. Since such deep networks require a lot of data, we increased our corpus of fonts(to fifteen as opposed to 8 earlier) to increase our learning set. We further added random noise (gaussian), rotations, zooming and horizontal/vertical shifting to increase variability and reduce chances of overfitting. This process of noise generation is different from our earlier case. Here we are using another amazing deep learning library : Keras[3], for all our training purposes. It preprocesses every batch of data while training and adds the above noise by itself. Our architecture was [Conv - Relu - Conv - Relu - Dropout]*2 - [Conv - Relu - Dropout] - FC layer - Softmax
The results that we got for our deep models were very similar to the ones that we were getting with the previous models. This suggests that the feature representations being learnt are very similar in both the cases. However, we did not get a chance to experiment more with the architecture and the hyperparameteres of the model.

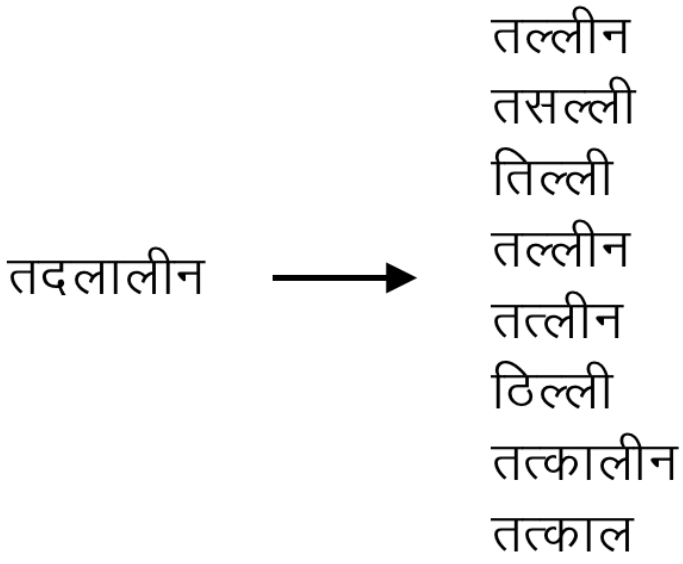

Figure 2.2: Example of suggestions returned by the dictionary

Figure 2.3: Correction in the predicted word after applying our heuristic

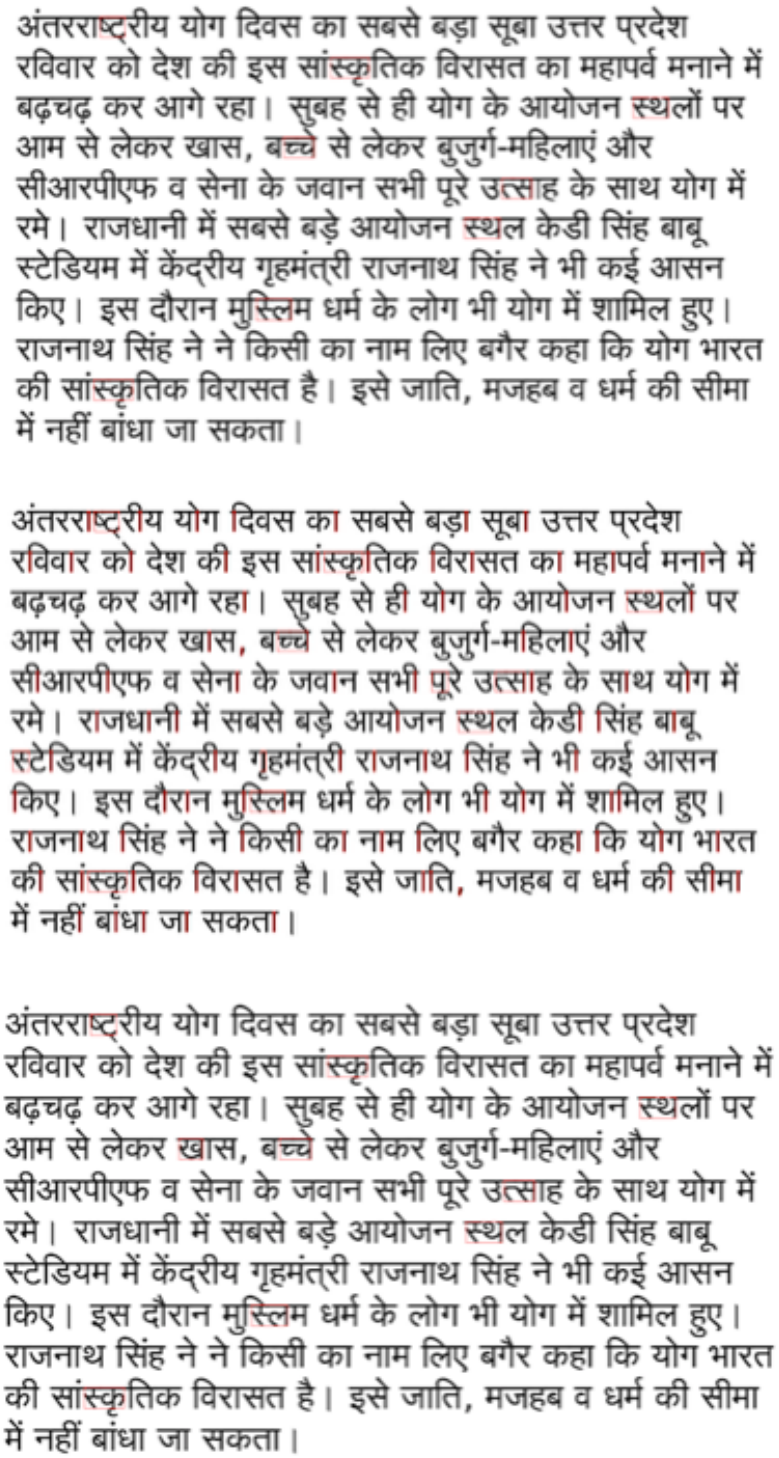

Figure 2.4: Comparison of how different models work for Jutakshar Detection. Model (b) on the top, followed by (a) and (c)

Chapter 3

Results

3.1 Jutakshar Detection

Document	Total Words	# Words containing Jutakshars	# Words containing Jutakshars and detected correctly	# Words incorrectly detected as containing Jutakshars
Doc 1	166	6	6	0
Doc 2	359	20	20	0
Doc 3	131	5	4	1
Doc 4	138	9	9	0

Table 3.1: Jutakshar Detection (Font 1)

Document	Total Words	# Words containing Jutakshars	# Words containing Jutakshars and detected correctly	# Words incorrectly detected as containing Jutakshars
Doc 1	145	6	6	0
Doc 2	218	4	4	0
Doc 3	138	4	4	0
Doc 4	167	7	7	2

Table 3.2: Jutakshar Detection (Font 2)

Document	Total Words	# Words containing Jutakshars	# Words containing Jutakshars and detected correctly	# Words incorrectly detected as containing Jutakshars
Doc 1	98	5	5	1
Doc 2	171	10	10	1
Doc 3	243	10	9	0
Doc 4	210	7	7	0

Table 3.3: Jutakshar Detection (Font 3)

Document	Total Words	# Words containing Jutakshars	# Words containing Jutakshars and detected correctly	# Words incorrectly detected as containing Jutakshars
Doc 1	72	3	3	1
Doc 2	99	3	3	1
Doc 3	114	8	8	1
Doc 4	133	5	5	0

Table 3.4: Jutakshar Detection (Font 4)

Document	Total Words	# Words containing Jutakshars	# Words containing Jutakshars and detected correctly	# Words incorrectly detected as containing Jutakshars
Doc 1	246	10	10	3
Doc 2	110	3	3	1
Doc 3	89	2	2	2
Doc 4	129	9	9	2

Table 3.5: Jutakshar Detection (Font 5)

3.2 Jutakshar Identification

Note that some words are not considered here since they were not present in the dictionary we used and hence have not been counted for computing accuracies

Document	Total Words	# Words containing Jutakshars	#Words containing Jutakshars and indentified correctly
Doc1	166	6	6
Doc2	357	18	17
Doc3	130	3	2
Doc4	138	8	3

Table 3.6: Jutakshar Identification (Font1)

Document	Total Words	# Words containing Jutakshars	#Words containing Jutakshars and indentified correctly
Doc1	143	4	2
Doc2	218	4	3
Doc3	138	4	0
Doc4	167	7	2

Table 3.7: Jutakshar Identification (Font2)

Document	Total Words	# Words containing Jutakshars	#Words containing Jutakshars and indentified correctly
Doc1	98	5	5
Doc2	170	9	6
Doc3	242	9	7
Doc4	209	6	2

Table 3.8: Jutakshar Identification (Font3)

Document	Total Words	# Words containing Jutakshars	#Words containing Jutakshars and indentified correctly
Doc1	71	2	1
Doc2	98	2	1
Doc3	113	7	6
Doc4	133	5	4

Table 3.9: Jutakshar Identification (Font4)

Document	Total Words	# Words containing Jutakshars	#Words containing Jutakshars and indentified correctly
Doc1	245	9	8
Doc2	109	2	2
Doc3	89	2	1
Doc4	128	8	6

Table 3.10: Jutakshar Identification (Font5)

3.3 Overall Document Accuracy

Here we have listed accuracies for three different models. The first model(I) is the model designed in [1], the (II) model includes our Jutakshar Detection model but with the same character recognition model as that of [1], and finally the (III) model includes the Jutakshar Detection model with the character recognition model trained on features extracted from the penultimate layer of a deep model (as discussed above). Note all punctuations have been removed while calculating accuracies.

Document	CER(%) (I)	WER(%) (I)	CER(%) (II)	WER(%) (II)	CER(%) (III)	WER(%) (III)
Doc 1	6.8	16.5	7.6	10.4	6.7	8.6
Doc 2	8.2	19.8	5.7	7.7	6.3	8.5
Doc 3	8.1	19.0	8.4	11.1	7.6	8.8
Doc 4	9.8	22.7	10.8	13.9	9.5	12.1
Overall	8.2	19.6	8.1	10.7	7.5	9.5

Table 3.11: OCR Accuracy : Comparing Models (Font 1)

Document	CER(%) (I)	WER(%) (I)	CER(%) (II)	WER(%) (II)	CER(%) (III)	WER(%) (III)
Doc 1	13.5	29.1	17.2	25.9	17.6	25.4
Doc 2	8.9	20.3	12.1	16.0	9.9	13.0
Doc 3	11.7	31.2	15.8	20.8	15.7	19.6
Doc 4	6.9	15.9	6.6	10.7	6.9	11.8
Overall	9.4	23.6	12.9	18.3	12.5	17.4

Table 3.12: OCR Accuracy : Comparing Models (Font 2)

Document	CER(%) (I)	WER(%) (I)	CER(%) (II)	WER(%) (II)	CER(%) (III)	WER(%) (III)
Doc 1	9.3	20.8	10.7	8.8	10.0	6.9
Doc 2	10.4	22.1	10.6	14.8	11.8	16.8
Doc 3	8.2	19.7	9.0	13.4	8.7	12.8
Doc 4	9.8	23.3	11.1	14.7	10.2	11.8
Overall	9.3	21.5	10.35	12.9	7.8	12.1

Table 3.13: OCR Accuracy : Comparing Models (Font 3)

Document	CER(%) (I)	WER(%) (I)	CER(%) (II)	WER(%) (II)	CER(%) (III)	WER(%) (III)
Doc 1	8.8	18.4	12.2	15.2	13.6	17.6
Doc 2	9.6	18.7	10.3	14.3	10.1	14.3
Doc 3	6.3	16.6	6.6	11.9	7.6	13.3
Doc 4	7.1	16.9	8.8	14	9.3	13.1
Overall	7.8	17.5	9.5	13.8	10.1	14.6

Table 3.14: OCR Accuracy : Comparing Models (Font 4)

Document	CER(%) (I)	WER(%) (I)	CER(%) (II)	WER(%) (II)	CER(%) (III)	WER(%) (III)
Doc 1	6.3	16.0	6.0	8.6	5.6	8.1
Doc 2	4.5	9.8	5.5	7.5	5.5	7.5
Doc 3	6.7	17.9	8.0	12.2	7.8	12.2
Doc 4	8.3	17.3	8.8	9.8	8.1	7.9
Overall	6.5	15.4	7.0	9.5	6.7	8.9

Table 3.15: OCR Accuracy : Comparing Models (Font 5)

3.4 Summary

We obtained encouraging results in detecting the presence of a Jutakshar in a word. While results obtained from the Jutakshar Identification model were reasonable, but lagged in some areas. Sometimes owing to a poor prediction of the word itself by the model, the dictionary suggested incorrect or no words at all. While in a few instances, either the dictionary itself gave incorrect words(words not part of the Hindi Language) or the correct word wasn't even part of the dictionary's vocabulary. However, the overall document accuracies still showed great improvement. Word Error Rates were significantly lower for all documents and across all fonts. The mean word error rate was 19.5% for (I) while 12.5% for model(III).

Chapter 4

Conclusion

We extended a Hindi OCR to specifically address the Jutakshar detection and identification issue. Classifiers were trained to identify words containing jutakshars using features extracted from a pretrained convolutional neural network and the width of the character box. The identified words were presented to a dictionary which provided a list of suggestions for the word. Our heuristic was then used to select the correct word from the list. The improvement in the overall document accuracies proves the effectiveness of our approach and provides motivation to try to resolve this issue further. We identified a few areas that could improve this system further:

- Our model relies heavily on the dictionary. Hence a more powerful dictionary would reduce quite a few errors
- The dataset can be made more representative by adding more fonts and possible jutakshars
- Two deepnet models could be trained - One for Jutakshars and the other for all the rest(since most of our errors were in correctly predicting characters other than the basic ones).

Bibliography

[1] Ankit Modi, Harish Karnick, B M Shukla, Optical Character Recognition for Devanagari with a Hindi Language Model, *http://172.28.64.70:8080/jspui/handle/123456789/15121*

[2] Jia, Yangqing, et al. 'Caffe: Convolutional architecture for fast feature embedding.' *Proceedings of the ACM International Conference on Multimedia. ACM,* 2014.

[3] Keras : *http://keras.io/*

[4] Dalal, N. and Triggs, B., Histograms of Oriented Gradients for Human Detection, IEEE *Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA*