Final Technical Report

Project Title: IndicView: Because language is no more a barrier (ILB)
Duration: 5 years (October, 2014 – September, 2019)
Sponsor: Google India Pvt. Ltd.
Principal Investigator: Dr. Pawan Goyal, CSE, IITKGP

Objectives

The primary aim of the project is to build an application that will be helpful in converting snippets of text written in Indic scripts to other languages. The end product for this project is a **mobile app as well as a web application that will extract Hindi text from images and convert it to English.**

This project is made keeping in mind the difficulties faced by non-Hindi speakers in understanding the text written in Hindi. It can be used to convert the manuscripts, news, books, magazines or billboards written in Hindi into equivalent English.

This app can be extended to include all Indian languages with an option to choose the source and target languages, and can help reduce the language gap among Indian languages.

Deliverables

- A web application where a user can upload photos (Hindi Text) and see the translated English text
- An android application where clicked photos (Hindi Text) will be translated and shown
- Testing under different scenarios like lighting conditions and computing the accuracies.
- Offline Functionality of the application
- The translated text should be of the same format as the source.

Workflow

Figure 1 shows the workflow of the proposed approach. We use the tesseract-ocr from Google as our main component but pre-process the image before giving it as input to Tesseract to improve the overall accuracy of the OCR. Once we obtain the OCR output, we do a post-processing step for error correction using language modeling. This text is then passed on to a Translation API and the translated text is shown to the user.

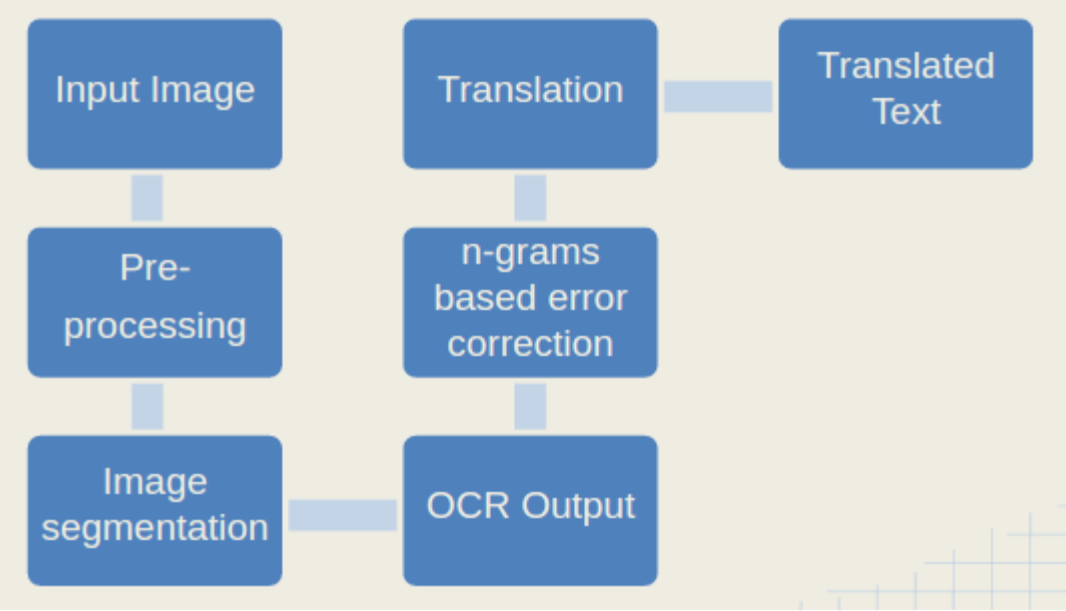

Figure 1: Workflow of the proposed approach

We now describe briefly various components used in our approach.

Pre-Processing of input image before passing it on to Tesseract

In this step, we tried out various pre-processing techniques that would be best suited for Hindi language. We implemented thresholding methods such as otsu, sauvola, multiscale sauvola, wolf, etc. Among these, the best results were obtained for sauvola thresholding. We, therefore, implement the sauvola thresholding algorithm and experimented with different kernel sizes. The kernel size for which performance is best is 15. Since the principle use case of this app is via the images taken from a camera,

we also implemented the skew, inverse perspective transform, and deblurring that will help the tesseract to better recognise the image.

During our experiments, we observed that tesseract gives better output if the image size is smaller and the image contains only a few words and less noise. So, for every image, we build bounding blobs for words and segment the image into a definite set of boxes with single words inside them. The boxes are ordered according to rows and they are passed one by one to tesseract. The output is put together to get the entire text as a paragraph. This gives far better results as compared to tesseract for images taken on phone as such images have a lot of noise.

Post-Processing of the Tesseract output

Once we obtain the output Hindi text from Tesseract, we run a post processing step to correct the output. Specifically, we apply a prime based algorithm which finds the nearest match to a word that is not there in the Hindi dictionary. On the top of this, we have applied a Non-parametric Bayesian Language Model (Teh 2006) in order to smoothen the final output. We did not use this in the android application due to time constraints.

Translating final output to English

In this step, we use Microsoft translator to translate the Hindi text to English. We used the API of the translator in both the android as well as web application. The android application is therefore an online application as the translator API requires internet connection.

Current Status

At present, we have both the web-application and the android application working. Below we describe these briefly.

Android Application

The android application provides the user the ability to check the output after each stage of computation. It has a very simple user interface. Users can click photos from

their camera and adjust the window upon the desired hindi text that they want to translate into English. The application uses a multi-threaded architecture for fast computation of the translated text.

At present, it is not compatible with the 64-bit processors due to initialization problem. We hope to add this in the near future. It might also be resolved due to future updates of OpenCV Manager. The android application can be downloaded from here.

Web Application

The web Application packs the complete workflow in a single application. The user can provide the image either from the device or via an url. We have built support for major image formats such as PNG, JPEG etc., which are generated from cameras. The web application can be accessed by visiting this link .

Evaluation Results

We evaluate the performance of our applications at various stages to establish that the proposed enhancements improve the overall performance. To evaluate the quality of OCR, we use levenshtein distance as our character-level accuracy checker. We also report the word level accuracies. To evaluate the quality of final English translation, we report the standard BLEU metric using ibleu.

Note that these evaluation require a ground truth data to compare against. We selected 74 images of Hindi text, each containing an average of 9.3 words. The input images used for testing can be found here. We prepared the ground truth manually for these images. For the final translation output, we used 25 images out of these 74, and provided manual translation as ground truth.

For the web application, we are reporting the average character level and word level accuracy of 74 images, each containing on an average 9.3 words and the BLEU score is being reported for 25 images out of these. For android application we are reporting the average BLEU Score of 25 images, each containing on an average 9.6 words.

Accuracy results as obtained on Web Application

Tables 1 and 2 report the character level and word level accuracies for the OCR output. We see that in comparison to the basic tesseract, our pre-processing step improves the performance by 13.7% and 14.9% respectively. We see that the post-processing step does not improve the character level accuracy but given an improvement at the the word-level.

Table 1: Character Level Accuracy for the OCR output

Method	Tesseract	Tesseract + Pre-Processing	Tesseract + Pre-Processing + Post-Processing
Accuracy (in%)	57.40	65.28 (+13.7%)	56.54

Table 2: Word Level Accuracy for the OCR output

Method	Tesseract	Tesseract + Pre-Processing	Tesseract + Pre-Processing + Post-Processing
Accuracy (in%)	47	54 (+14.9%)	54.40

Table 3 reports the BLEU scores for the final translation. Note that to put these values in perspective, we also report the BLEU score, when the Ground Truth Hindi text is directly given as input to the translation API. For our experiments, this serves as a good upper bound on the translation quality. We see that if we directly input the output of Tesseract to the translation API, BLEU score is only 1.7. On the other hand, our web application gives a BLEU score of 3.14, an improvement of 84.7% over the basic model.

Table 3: BLEU Metric Scores for the final translation

Method	Translation API with Ground Truth Hindi text	Tesseract + Translation API	Final Web Application
BLEU score	6.18	1.7	3.14 (+84.7%)

Accuracy Results on the Android Application

Tables 4 and 5 report the BLEU metric scores for the Android App with 2 different Mobiles. Mobile 1 and Mobile 2 represent two different mobiles with different camera apertures, and quality. Both mobiles had an 8MP camera, but Mobile 2 included auto-focus feature. We test our app under two different lighting conditions. Lighting condition 1 indicates the presence of enough light, either through ambient light (day clicked pictures) or through flash. Lighting condition 2 represents that these conditions were not satisfied.

We see that while the auto-focus feature of camera 2 resulted in slightly better accuracy on the plain tesseract, our method is robust to the camera as well as lighting differences. In general, the BLEU score obtained using our App is much better than that obtained using the basic model.

Table 4: BLEU Metric Scores for the Android App, when tested for Mobile 1

Method	Tesseract-Lighting Condition 1	Tesseract-Lighting Condition 2	Android Application - Lighting Condition 1	Android Application - Lighting condition 2
BLEU score	1.4	1.42	2.86	2.89

Table 5: BLEU Metric Scores for the Android App, when tested for Mobile 1

Method	Tesseract-Lighting Condition 1	Tesseract-Lighting Condition 2	Android Application - Lighting condition 1	Android Application - Lighting condition 2
BLEU score	1.5	1.51	2.88	2.90

Conclusions

As is clear from the accuracy results for the Web and android application, by using our proposed enhancements, we were able to improve upon the scores from the state-of-the-art tesseract library. The results, however, leave a lot of scope for further improvements in the OCR quality. Note that the translation API can easily be replaced by another such module and the target language can also be varied without much changes to the proposed method.