Bangla Character Recognition for Android Devices

Aparajita Chowdhury
Dept. of Computer Science & Engineering
BRAC University
Dhaka

Abu Foysal
Dept. of Computer Science & Engineering
BRAC University
Dhaka

Shafiqul Islam
Dept. of Computer Science & Engineering
BRAC University
Dhaka

ABSTRACT

The main target of the project was to build an Android application that can extract text from any image that contains Bengali characters and convert it into an editable document. There were a few limitations in existing systems which could be improved further. To recognize more characters and joint letters, it was decided to work on decreasing the rate of error to preserve more texts. Tesseract (v3.03) was used to recognize the characters which utilizes Leptonica Image Processing library to process image and extracting data from the image. Joint letters, dangerous ambiguity and contrast issues were handled to increase efficiency. A record of the analyzed data and overall progress were kept for future scopes of improvement.

Keywords

Optical Character Recognition (OCR), Bangla language, Android, Tesseract, Leptonica.

1. INTRODUCTION

Optical Character Recognition (OCR) is a field of research in Computer Science that conducts the task of reading text in image format and converting that into a text form that can be further modified in the computer. It turns out from the research that lots of works have been done for OCR purpose. But none of them are significant enough for Bengali to meet the need. As a result, an initiative was taken for developing an Android application for Bengali OCR to preserve Bengali printed text documents by making softcopies as text files.

2. RELATED WORKS

While planning and making progress for the project, many research papers related to OCR were gone through. Since the number of analysis on Bangla OCR was not sufficient enough, studying the papers for English and Hindi OCR fruitful information were found for making progress. Smith [1], helped with his analysis of recognition of Tesseract on damaged fonts and the number of fonts to train. Moreover, Omee discusses in their paper about case sensitivity and "Matra" (line over characters) of Bangla characters [2]. In addition to that Hasnat [3] in his paper discusses about the number of attempts undertaken to resolve OCR for printed texts using HMM (Hidden Markov Model). Zaman [4], however, mentioned in their research about languages, except Bengali. Chowdhury [5], on the other hand attempted for Bangla OCR for non-portable desktop version. There were many attempts on hand written texts such as Rakshit's [6].

3. SYSTEM ARCHITECTURE

3.1 Tesseract OCR

From the observation, Tesseract was found to be the most accurate open-source OCR engine. Tesseract can read a wide variety of image formats and convert them to text in over 40 different languages. However, Tesseract was originally designed to recognize English text only. To deal with other languages and UTF-8 characters, such as Bengali, several efforts have been made to modify its engine and the training system [7]. The system structure itself needed to be changed to make Tesseract able to deal with languages other than English. Tesseract 3.0 can handle any Unicode character. However, there were limits as to the range of languages that Tesseract will be able to successfully detect. Therefore, adequate actions has been taken to make sure that Bangla language gets recognized by Tesseract.

Tesseract 3.01 added top-to-bottom languages, and Tesseract 3.02 added Hebrew (right-to-left). Tesseract can currently handle complex scripts like Arabic with an auxiliary engine called cube. However, cube, is not yet equipped to detect the Bangla language. Additionally, it includes "unicharset" to make multi-language handling easier. This function aided the use of four different Bangla fonts to train and detect characters. Though Tesseract is slower with a large character set language (like Chinese), but it seems to work nonetheless. Tesseract also takes more time to detect Bangla character compared to detect English characters. However, it can still detect Bangla characters which is the main purpose of the research.

Tesseract 3.03 added new training tool text2image to generate box/tiff files directly from text. It also has support for PDF output with searchable text. However, the text is only searchable, as it will be in PDF format and cannot be edited. Tesseract v3.03 (rc) is currently the latest available Tesseract engine, therefore, this engine was used in the research.

Initially, there was a desktop version programmed in the C programming language. It was only capable of detecting text from images saved on the computer. However, using tess-two library Tesseract and Leptonica Image Processing Library can be used on the Android platform. Tess-two is a fork of the Tesseract Tools for Android that provides the ability to utilize the OCR engine on an Android device. The Tesseract Tools for Android is a set of the following three features:

- Android API
- Tesseract OCR engine
- Leptonica Image Processing Library.

The tess-two comes up with the tools for compiling and running both Tesseract and Leptonica Image Processing Library on the Android OS.

3.1.1 Skewness and Ambiguity:

Among all problems related to OCR, one of the most significant is the ambiguity. It tends to confuse similar letters, for example "o" and "0" [8]. Moreover, it will be similarly perplexing for OCR to detect characters on dark backgrounds [8]. Bengali is much more prone to detection failure having large set of joint letters.

Image Skewness was also an issue to be noticed in case of OCR. Skewness refers to the tilt in the bitmapped image of the scanned document image for Optical character Recognition [9]. Pal [10], used the approach of recognizing the character shapes by a combination of template and feature matching approach. Images are digitized by flatbed scanner and subjected to skew correction, line, word and character segmentation, simple and compound character separation. They have used a feature based tree classifier for simple character recognition. Chaudhuri, in their proposed model [11], used document digitization, skew detection, text line segmentation and zone separation, word and character segmentation, character grouping into basic, modifier and compound character category for both Bangla and Devnagari (Hindi). Their system showed a good performance for single font scripts, printed on clear documents. Smith [5] mentions in his paper, an important part of any document recognition system is detection of skew in the image of a page. Their paper presents a new, accurate and robust skew detection algorithm based on a method for finding rows of text in page images. Not all image texts are uniformly symmetrical. To get the skewness of an image it can be converted to a binary file after making a grayscale from the source image. Next, from mathematical formulae of distribution, the angle of skewness is measured. Sarfraz [12] discusses in his paper, an input image needs to be normalized and converted into a format accepted by the OCR system. The OCR systems typically assume that the documents were printed with a single direction of the text and that the acquisition process did not introduce a relevant skew. However, practically this assumption is not very strong and printed document could be skewed at some angle with the horizontal axis.

- A perfect normal distribution provides tapering equally from both sides maintaining symmetry.
- In a left skewed distribution left side is longer than the right side tail.
- In a positively skewed distribution right hand side tail is longer than the left one.

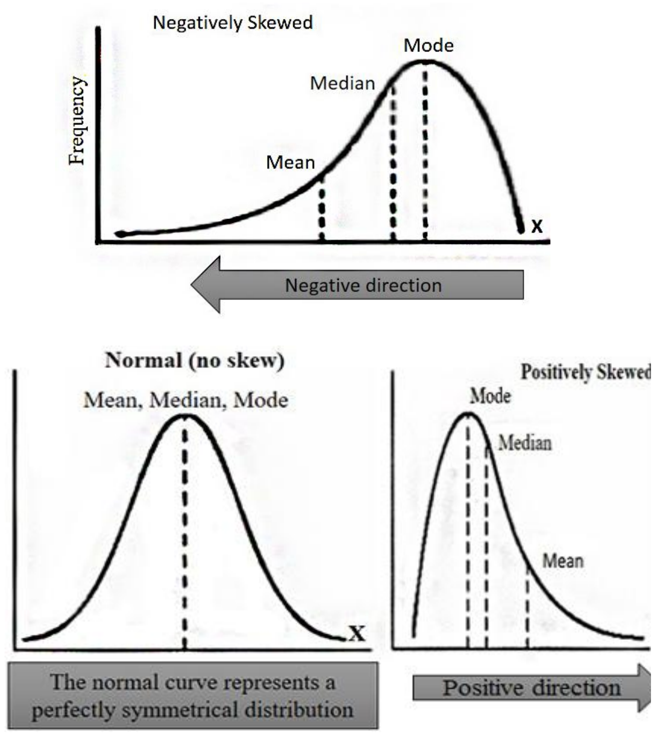


Figure1: Skewness

In the figure above, one is normal Bell Curve of normal distribution where it turns out that the average and the peak are equal. In top the negatively skewed is also called a left skewed graph. Similarly, positively skewed is a right skewed graph. There are several methods of calculating skewness.

Formula for calculating skewness:

Pearson's Formula:

$$\frac{(mean - mode)}{\sigma}$$

Table 1: Pearson's Formula

3.2 Development Environment

This project uses the tess-two library to incorporate Tesseract and Leptonica Image Processing Library for using in any Android Platform. To use tess-two, at first, it was built on the Linux based operating system. For this Ubuntu OS was used. Tess-two contains an Android library project that provides a Java API for accessing natively-compiled Tesseract and Leptonica APIs. Android SDK and Android NDK were used to build the project in Ubuntu. However, after completing building the application, the platform was shifted to Windows Operating System.

The main advantage of this shifting was, some Windows based softwares like Serak Tesseract Trainer and QT Box Editor could be used for working with multiple files in batch. QT could be used to edit and create box files very easily. By using Serak, the scripting of Tesseract on TIFF/box file pairs could be automatized. In Ubuntu, all the file names of the training images and box files in the terminal had to be manually entered to run Tesseract on them. The number of files is quite large considering the fact that a huge data set was being trained. By using Serak, the overall effort is minimized.

The Tesseract engine with tess-two library have been successfully installed and run using the following configurations:

1. Ubuntu 14
2. Windows7/8.1/10
3. Android 2.2 +
4. Tesseract v3.03
5. Leptonica Image Processing Library v1.72
6. Four trained data file for four different types of Bangla language.

3.3 Softwares Used for Training

3.3.1 JTessBoxEditor

jTessBoxEditor v1.4 is a Java based software created by a Vietnamese company, VietOCR. It is a box file editor and trainer for Tesseract OCR, which provides the functions to edit the box data in both Tesseract 2.0x and 3.0x formats and full automation of Tesseract training. It can read images of common image formats, including multi-page TIFF. This program requires Java Runtime Environment 7 or later to operate. This program was used by Gajoui and Banerjee in their research to train Tesseract [13], [14], so it was decided to use it as well.

The editor mode of jTessBoxEditor requires the TIFF/Box files as input. The images to be used in training should be of 300 DPI and 1 bpp (bit per pixel) black and white or 8 bpp grayscale uncompressed TIFF format. For box files, the file needs to be encoded in UTF-8 format which can be generated by Tesseract executables with appropriate command-line options or they can also be created using the built-in TIFF/Box Generator of jTessBoxEditor.

For the project only TIFF/Box Generator function has been extensively used. For a given input UTF-8 text file, the generator produces an image in TIFF format along with a Box file. The box contains the mapping of the characters that were in the text file. The generated image is, depending on anti-aliasing mode enabled, a binary or 8-bpp grayscale, uncompressed multi-page TIFF with 300-DPI resolution. Noise has been included in the training image, so that it results in better trained data. Letter tracking, or spacing between characters, can be adjusted to eliminate bounding box overlapping issues. Overlap in the boxes causes huge problems for Bengali characters. As it is clear that, the characters in the Bangla alphabet are not uniform in shape. So creation of the character boundaries in the box files had to be done very carefully.

3.3.2 QT Box Editor

The QT Box Editor is a tool for adjusting tesseract-ocr box files. The aim of this project was to provide an easy and efficient way for editing regardless of file size. The QT Box Editor is a successor of the tesseract-gui project that is no longer in development. This software is used to edit already created box files. This software has been instead of jTessEditor because QT is much more sophisticated. Editing boxes in bulk is a very time consuming job. However, with QT the amount of time needed to process each box file is reduced significantly. Furthermore, QT has some very useful functions like insert, merge, split and delete right at the tip of the mouse. However, multipage TIFF is not supported yet and due to image quality and space the use of TIFF with compression or PNG was confirmed. QT was also used by Banerjee in their research [14].

3.3.3 Serak Tesseract Trainer v0.4

Serak Tesseract Trainer is a front end GUI for Training Tesseract 3.02. Serak has basically been used to automate the training process in the Windows OS Environment. Serak has the feature to create traineddata files using only TIFF/Box file pairs. Serak was used at the very end of this project to combine all the TIFF and box files and create a traineddata for the Tesseract engine. It is very useful when dealing with a large number of files. In this research, there was a pretty big character set to deal with. So, manually executing all the process and functions in creating a traineddata file is very inefficient and would require a lot of effort.

After successfully creating the traineddata file, it was fed to Serak and used Serak's OCR Mode to run Tesseract on images to detect characters. Using Serak's OCR Mode one is able to test and verify the integrity of the data set trained using the TIFF and box files.

4. WORKFLOW

The development process of Dristee OCR was conducted in two stages. First part included training and second part involved implementation of the Android application which uses the traineddata file and recognizes Bengali text and converts it into a searchable and editable document.

4.1.1 Training Dristee OCR

This project is completely dependent upon the quality and quantity of the text that was used for training. While training, it had to be ensured that there were no mistakes in the character sets. This is because what one may think of as a slight error, may end up significantly decreasing the accuracy of the application. Next, the character set itself is vital, as all the existing Bengali characters had to be covered.

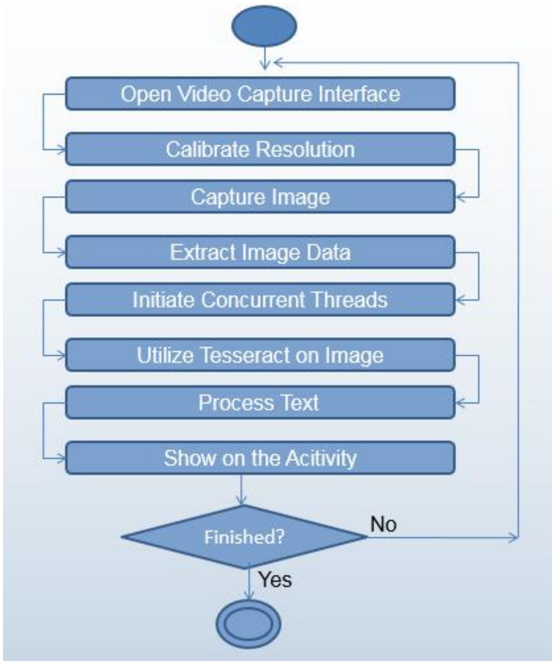
The number of Bengali characters gets very large, if the joint letters are considered for training. As this research is about trying to increase the accuracy, all possible joint letters was included in the training data. After preparing the text data for training, they were converted into images using the software jTessBoxEditor. Four sets of training data file for four fonts were created. Consequently, using QT Box Editor necessary changes were made for the box files, making sure that the characters were correctly mapped inside the boxes.

Additionally, it is essential that there are no overlapping boxes in the training set, as it may create a shapeclustering error in later stages of training. Finally, after successfully creating the box files, Serak Tesseract Trainer was used to automatize the training process and combine the traineddata file using the box files.

4.1.2 Executing Dristee OCR

As the project was about developing a real time system for character detection, the video capture interface of the camera on our Android Phone was used. The calibration of the resolution was done so that it can avoid any kind of distortion like parallax error. After that, some concurrent internal processes were executed like capturing the image and preparing it for the OCR. Simultaneously, Leptonica Image Processing Library was used and the data was handed over to Tesseract for character recognition.

Ultimately, when the recognition is done by properly matching with box files and corresponding Unicode Character, the result will be converted to text format and shown in the activity window of the application on the Android device. The text file can be saved in the device for future use if necessary.



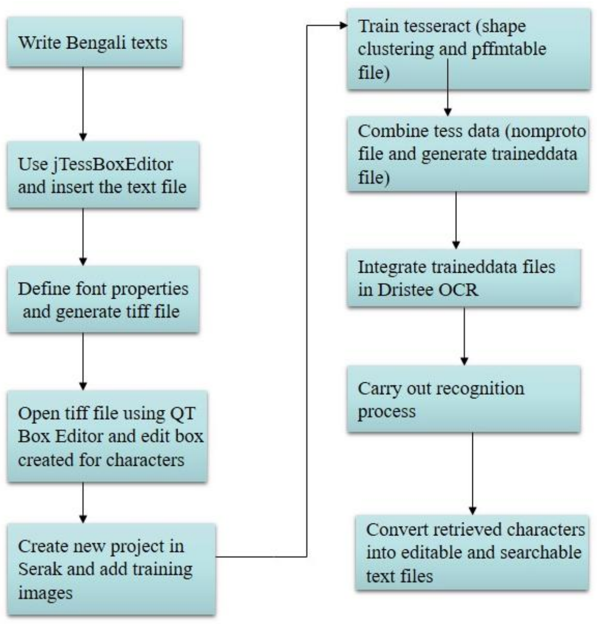

Write Bengali texts
Use jTessBoxEditor and insert the text file
Define font properties and generate tiff file
Open tiff file using QT Box Editor and edit box created for characters
Create new project in Serak and add training images
Train tesseract (shape clustering and pffmtable file)
Combine tess data (nomproto file and generate traineddata file)
Integrate traineddata files in Dristee OCR
Carry out recognition process
Convert retrieved characters into editable and searchable text files

Figure 2: System Workflow

5. SYSTEM IMPLEMENTATION

5.1 Generating Image Files from Text Files

To start the training process of the application, the images were needed to train with. So, good quality image files had to be generated from text. For this purpose, jTessEditor was used. By using the TIFF/Box Generator method, text can be easily converted to images in TIFF format directly. TIFF format image is recommended for training Tesseract. The software jTessEditor also creates Box files of the characters given in the text files. Here, the input as txt file for raw text in the text box was provided. A total of four fonts were trained, AdorshoLipi, Kalpurush, Nikosh and SolaimanLipi. In addition to that "noise" was added in jTessBoxEditor, to artificially add noise to the training images. Adding noise increases the detection level of the OCR. For this research, noise value of 5 and the font size 12 were used. Later on, data set was increased using different font sizes such as, 14, 18, 24 and 36.

It has been found that using a variety of font sizes help increase the accuracy of the OCR. Initially, font size 12 was only used for all the texts. However, using the traineddata, created using these images resulted in poor accuracy. Therefore, it was decided to increase the font size and create new sets of training images and box files. With these new traineddata, it was able to increase the recognition to some extent.

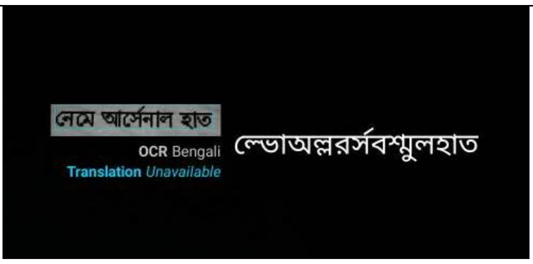


Figure 3: Detection using only font size 12

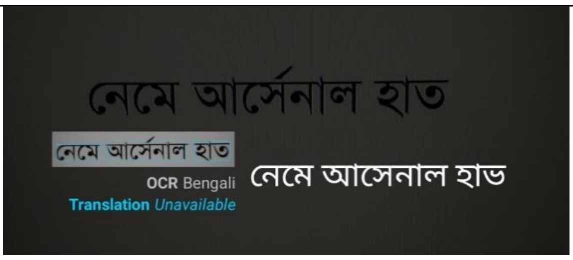


Figure 4: Detection after using multiple font sizes

5.2 Types of Characters Used for Training

In this research, all the characters in the Bengali alphabet were tried to cover; however, Bengali numbers have not been included in the final data set. During the testing phase, it was found that including number increases the ambiguity of the characters. Several other issues were addressed by Datta [15] in his paper. While most numbers were harmless except for "২" and "৩", the OCR was quite frequently confused between these two numbers. It was also seen to be confusing অ and ত with ৩ on several occasions. This results in decreased overall accuracy. Moreover, it was chosen not to train the vowel diacritics (া ি ী ু ৃ ে ৈ ো ৌ...) individually either. Training with these characters also affects the detection level of the OCR. This fact was already confirmed by Zaman [2] in their paper. So, it was concluded that it would be better to associate these diacritic characters with consonants forcefully and train them for the OCR application. More information on the properties of different Bengali scripts can be found at [16]. Lastly, the whole research concentrated on the successful detection of conjunct consonants. No work has been done to detect these joint letters precisely.

ই, ঈ -> ২
অ, ত-> ৩

Table 2: Numerical Ambiguities

Bangla Vowels	অ আ ই ঈ উ ঊ ঋ এ ঐ ও ঔ
Bangla Consonants	ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড ঢ ণ ত থ দ ধ ন প ফ ব ভ ম য র ল শ ষ স হ ড় ঢ় য় ৎ ং ঃ ঁ
Bangla Conjunct Consonants	ন্ঠ ন্ত স্ক স্ক্র ষ্ট ক্ষ ষ্ক শ্ল হ্ল স্ল ড্ড ঞ্ঝ...
Bangla Consonants with diacritics	কা খা ঠি ধী চু কূ জূ ঢে খৈ গো ঘৌ...

Table 3: Types of characters trained

A total number of 12,191 individual Bengali characters were trained for the Dristee OCR application. This includes all four fonts which consists of all possible Bengali characters. The majority of this data set consists of conjunct consonants as they are huge in number, 8971 to be precise.

5.3 Creating Box Files

After converting the text into images, box files had to be created for them. Box files are simply the mapping of the characters in the images. QT Box Editor was used for creating and editing the box files. It is required to merge characters like ক, া into কা as the vowel diacritics are not desired to be separate. Doing this in jTessBoxEditor is troublesome, therefore QT was used.

The Box files contain the characters in the image in Unicode format as well as their corresponding "Left, Bottom, Right, Top" position with respect to the dimensions of the images.

5.4 Processing Box Files

After all the box file pairs were created, Tesseract is needed to be run on each of the images and corresponding box files. For each of the images and box file pairs, the following command must be executed:

For Windows Platform Exclusively:

```
tesseract [lang].[fontname].exp[num].tif
[lang].[fontname].exp[num] box.train

tesseract ben.solaimanlipi.exp0.tif ben.solaimanlipi.exp0
box.train
```

For All Platforms:

```
tesseract [lang].[fontname].exp[num].tif
[lang].[fontname].exp[num] box.train.stderr

tesseract ben.solaimanlipi.exp0.tif ben.solaimanlipi.exp0
box.train.stderr
```

However, it is not feasible to run this command for a large number of training images. It takes both effort and time to manually execute all the commands. Therefore, Serak Tesseract Trainer was used to automate the task. Serak runs this command relentlessly until all the images and box files are processed.Executing this command line creates a [lang].[fontname].exp[num].tr file for every pair. Eg. ben.solaimanlipi.exp0.tr.

5.5 Computing the Character Set

Next, a unicharset data file must be created, which lets Tesseract know the set of possible characters it can output. For this the unicharset_extractor program is needed to be used on the box files generated above. However, unlike previous processes, it is not needed to type the same command over and over for every box file. The name of the box files can be appended separated by a space.

```
unicharset_extractor lang.fontname.exp0.box ....
lang.fontname.expN.box

unicharset_extractor ben.solaimanlipi.exp0.box ....
ben.solaimanlipi.expN.box
```

5.5.1 Setting the Unicharset Properties

A new tool and set of data files in 3.03 allow the addition of extra properties in the unicharset, mostly sizes obtained from different fonts.

```
training/set_unicharset_properties -U input_unicharset -O
output_unicharset --script_dir=training/langdata
```

5.5.2 Font Properties

Subsequently, the font_properties file is needed to be generated. This file contains all the information about the style of the text. It controls the style the output receives after the font is recognized. The font_properties file is a text file specified by the -F filename option to mftraining.

Each line of the font_properties file is formatted as follows:

<fontname> <italic> <bold> <fixed> <serif> <fraktur> Here, <fontname> is a string naming the font and <italic>, <bold>, <fixed>, <serif> and <fraktur> are all simple 0 or 1 flags indicating the respective state of the font.
Eg. solaimanlipi 0 0 0 0 0, means that the font name is solaimanlipi with no styling present.

6. FEATURES

6.1 Torch and Exposure

From a few observations, it had been seen that the OCR result in lower light in case of printed text is not satisfactory. A bit of research was done and monochrome and colored light model from graphics was discovered to fix this issue. Finally, it was concluded that using the camera flash will help develop the result to a great extent. Screen images use emission based technology. On the other hand, printed document detection needs to be implemented with reflection based technology. In the project a package called bracu.ac.bd.ocr.camera was introduced, here the class CameraConfigurationManager.java was included. This class utilizes the methods doSetTorch and setFlashMode to enable flash.

Exposure is another issue to increase the quality of the image before detection and compromise contrast. This is dealt by CameraManager.java. Consequently, three exposure levels called low, medium and high are given as option to the user. The method setExposureCompensation is used to apply the exposure.

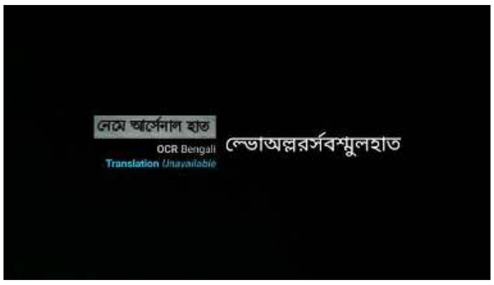


Figure 5: Before Exposure Control

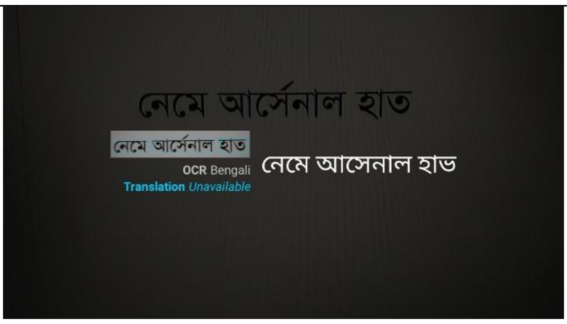


Figure 6: After Exposure Control

From the test images above, it is observed that the exposure of the camera plays an important role in image quality. Hence, efficient detection of the characters depends on it. However, it must be acknowledged the fact that not all cameras have the same exposure level. Therefore, exposure level has been margined using the minimum and maximum exposure possible by the camera. Next, this information was manipulated in setting three exposure levels, low, medium and high.

7. RESULTS

The traineddata had been tested in both desktop and mobile version. Precision and recall had been used to calculate the accuracy of this OCR system [17]. Here, precision is the fraction of how many retrieved results are correct and recall gives an estimated fraction of how many positives is returned by the model. At first, the Android application was tested with AdorshoLipi and Nikosh without any exposure control. The result was found to be erratic as the light was not sufficient due to the fact that the pictures need to be taken from a close distance from the camera. Therefore, the flashlight on the Android phone was used to increase the quality of light. Moreover, manual selection of exposure setting was also added to help obtain better results. Later, for the fonts Kalpurush and SolaimanLipi these conditions were handled and the results are more satisfactory than the previous two fonts.

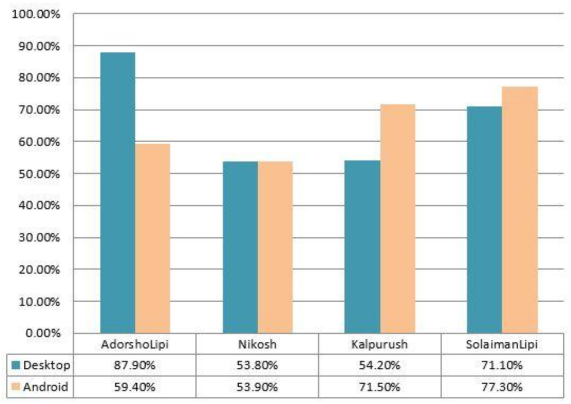
Table 4: Desktop Version

Font	Precision	Recall
AdorshoLipi	87.9%	87.7%
Nikosh	53.9%	55.7%
Kalpurush	54.2%	64.8%
SolaimanLipi	71.1%	75.2%

During the testing, an average precision of 66.8% and recall of 70.9% for the desktop version were found. For the mobile version the average precision was 65.5% and recall was 70.0%.

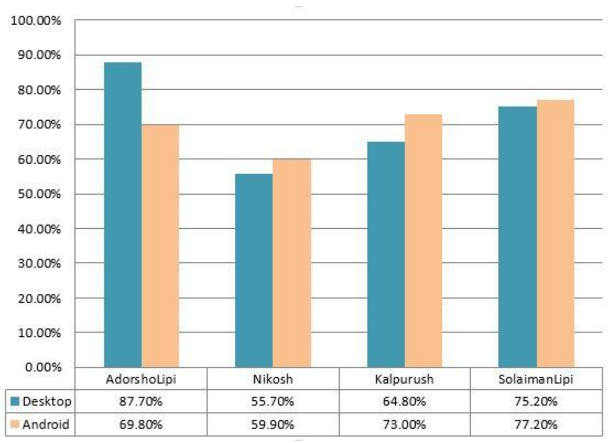
Table 5: Android Version

Font	Precision	Recall
AdorshoLipi	59.4%	69.8%
Nikosh	53.9%	59.9%
Kalpurush	71.5%	73.0%
SolaimanLipi	77.3%	77.2%



	AdorshoLipi	Nikosh	Kalpurush	SolaimanLipi
■ Desktop	87.90%	53.80%	54.20%	71.10%
■ Android	59.40%	53.90%	71.50%	77.30%

Figure 7: Precision



	AdorshoLipi	Nikosh	Kalpurush	SolaimanLipi
■ Desktop	87.70%	55.70%	64.80%	75.20%
■ Android	69.80%	59.90%	73.00%	77.20%

Figure 8: Recall

8. CONCLUSION

From the extensive research, literature review and related work, various approaches of handling the shortcomings of character recognition were observed. However, there still persist some limitations in the current research, which could be further improved by deploying other existing methods and applications. For example, improving accuracy in case of joint letters in hand-written texts and cursive text. There are also scopes to improvise different algorithms for example the new method of feature extraction procedure called zoning and template matching combined mentioned by Arif [8]. There are hardly any attempts to create a lexicon of Bengali language [20]. The few that exists lack colloquial language and proper noun which are expanding day by day. Therefore, there is still scope left to build a lexicon for Bengali. There are also scopes like standardizing encoded language, creation of lexicon for Bengali characters and camera positioning issues. Moreover, it would be mesmerizing to integrate with technologies like Google Glass for real time detection. In addition to that, experimenting on the contrast issue can also boost up accuracy in recognition.

In short, the intention was to improve the existing systems such that the accuracy level could get closer to 100%. The main objective was trying to make Tesseract recognize Bengali fonts. Overall, the research was successful in order to make Tesseract more precise by training space as well as the joint characters. As a result, a better solution was obtained. Along with the joint characters, different fonts were introduced to cover more variations. Finally, the research was concluded with the complete implementation of a portable Android application for Bangla OCR. This application is user friendly and efficient in detecting Bengali characters from images and converting them to editable text files.

9. ACKNOWLEDGMENTS

The authors thank CSE Department of BRAC University for providing required resources to accomplish this research.

10. REFERENCES

[1] Smith, R. (2007). An Overview of the Tesseract OCR Engine. Proc. of 9th ICDAR 2007, Curitiba, Paraná, Brazil. (pp. 629-633). IEEE Explore.

[2] Omee, F. Y., Himel, S. S., & Bikas, M. A. N. (2011). A Complete Workflow for Development of Bangla OCR. International Journal of Computer Applications, 21(9).

[3] Hasnat, M. A., Habib, S. M. M., Khan, M. (2008). A High Performance Domain Specific OCR for Bangla

Script. Novel Algorithms and Techniques in Telecommunications, Automation and Industrial Electronics. (pp. 174-178).

[4] Zaman, S. M., & Islam, T. (2012). Application of Augmented Reality: Mobile Camera Based Bangla Text Detection and Translation. BRAC University.

[5] Chowdhury, M., T., Islam, M., S., Bipu, B., H. (2015). Implementation of an Optical Character Recognizer (OCR) for Bengali language. BRAC University.

[6] Rakshit, S., Ghosal, D., Das, T., Dutta, S., Basu, S. (2009). Development of a Multi-User Recognition Engine for Handwritten Bangla Basic Characters and Digits. Int. Conf. on Information Technology and Business Intelligence.

[7] Hasnat, M., A., Chowdhury, M., R., Khan, M. (2009). Integrating Bangla script recognition support in Tesseract OCR. BRAC University.

[8] Patel, C., Patel, A., & Patel, D. (2012). Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study. International Journal of Computer Applications, 55(10).

[9] Aithal, P., K., Acharya, U., D., Siddalingaswamy, P., C. (2013). A Fast and Novel Skew Estimation Approach using Radon Transform. International Journal of Computer Information Systems and Industrial Management Applications (5). (pp. 337-344).

[10] Pal, U., Chaudhuri, B., B. (1994). OCR in Bangla: an Indo-Bangladeshi language. Proc. of ICPR, Jerusalem, Israel. (pp. 269-274). IEEE Explore

[11] Chaudhuri, B., B., Pal, U. (1997). An OCR system to read two Indian language scripts: Bangla and Devnagari (Hindi). Proc. of 4th ICDAR. Ulm, Germany. (pp. 1011-1015). IEEE Explore

[12] Sarfraz, M., Zidouri, A., Shahab, S.A. (2005). A novel approach for skew estimation of document images in OCR system. International Conference on Computer Graphics, Imaging and Vision: New Trends. (pp. 175-180). IEEE Explore.

[13] Gajoui, K., E., Ataa-Allah, F., Oumsis, M. (2015). Training Tesseract Tool for Amazigh OCR. Recent Researches in Applied Computer Science. Proc. of 15th International Conference on Applied Computer Science (ACS15), Konya, Turkey. (pp.172-179). WSEAS Press.

[14] Banerjee, S. (2012). A Study on Tesseract Open Source Optical Character Recognition Engine. Jadavpur University. Retrieved December 13, 2015, from: http://dspace.jdvu.ac.in /handle/123456789/27793

[15] Datta, S., Chaudhury, S., and Parthasarathy, G. (1992). On Recognition of Bengali Numerals with BackPropagation Learning. IEEE International Conference on Systems, Man and Cybernetics (pp. 94-99). IEEE Explore.

[16] Abdullah, A., Khan, M. (2007). A Survey on Script Segmentation for Bangla OCR. BRAC University.

[17] Manning, C., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. Cambridge, Mass. MIT Press.

[18] Arif, S., R. (2007). Bengali Character Recognition using Feature Extraction. BRAC University.

[19] Hayder, K. (2007). Research Report on Bangla Lexicon. BRAC University.