Implementation of an Optical Character Reader (OCR) for Bengali Language

Muhammed Tawfiq Chowdhury, Md. Saiful Islam, Baijed Hossain Bipul and Md. Khalilur Rhaman
School of Engineering and Computer Science, BRAC University
Dhaka, Bangladesh
{m.tawfiq.c, ranabrac.islam, baijed.hossain5}@gmail.com, khalilur@bracu.ac.bd

***Abstract*— Optical Character Recognition (OCR) is the process of extracting text from an image. The main purpose of an OCR is to make editable documents from existing paper documents or image files. Significant number of algorithms is required to develop an OCR and basically it works in two phases such as character and word detection. In case of a more sophisticated approach, an OCR also works on sentence detection to preserve a document's structure. It has been found that researchers put lots of efforts for developing a Bengali OCR but none of them is completely error free. To take this issue in consideration, the latest 3.03 version of Tesseract OCR engine for Windows operating system is used to develop an OCR for Bengali language. Moreover, 18110 characters and 2617 words are used to make the OCR's library. In this research, 'Solaimanlipi' font and 200 input files are used to test the accuracy of OCR. It is found that for clean image files, the accuracy of the software is as high as 97.56%. It is to be noted that accuracy is measured as the percentage of correct characters and words.**

Keywords — OCR; Tesseract; Bengali

I. INTRODUCTION

In our day to day life, we often need to reprint text with modification. However, in many cases, the printable document of the text does not remain available for editing. For example, if a newspaper article was published 10 years ago, it is quite possible that the text is not available in an editable document such as a word or text file. So, the only choice remains is to type the entire text which is a very exhaustive process if the text is large. The solution of this problem is optical character recognition [1]. It is a process which takes images as inputs and generates the texts contained in the input. So, a user can take an image of the text that he or she wants to print, feed the image into an optical character recognition (OCR) software and then the software will generate an editable text file for the user which is amendable. This file can be used to print or publish the required text.

For international languages such as English, OCR has been developed quite a long ago and are available in different dimensions. However, for Bengali language, there has not been much work done although in recent time, some projects have been implemented. However, none of them are fully accurate. There have been detached works with no integration. It is not also easy to find much information about developing an OCR for Bengali. Different projects have been implemented in different methods. Some developers used their own algorithms to develop an OCR while some others used existing OCR engines to make it. It is not quite easy to develop an OCR for Indic languages like Bengali because of different complexities. For example, Bengali has diverse types of characters and they sum to a very huge number. The inter resemblance among the characters makes it even tougher to maintain the accuracy as the OCR may misjudge one character for another. The total number of characters also makes the execution time longer as the scanning process of OCR goes through a very large data set. We preferred to work on an OCR engine for our thesis project. This engine called 'Tesseract' is well tested. Though there are some limitations, we trained the engine for Bengali for a very intricate and large character set and the performance of the trained OCR is satisfactory.

II. CHARACTERISTICS OF BENGALI SCRIPT

Bengali has a very complex script pattern. Not only it has vowels and consonants but also it has combined characters which are composed of several other characters and special characters [3]. Each word in Bengali scripts is formed of a number of characters connected by a horizontal line called 'Maatra' or head-line at the top of the characters but some characters are exception to this [4]. Bengali has a few basic script features. These have been shown below.
a) Style of Bengali writing is from left to right.
b) Bengali does not have the variation of upper and lower case.
c) There are short forms of vowels of Bengali named 'kar' such as ◌া, ◌ি, ◌ু and consonants named 'Fola' such as ম্ল, ন্য.
d) In a solitary syllable of a word, a few consonant characters may join to constitute a compound characters.
e) There are few characters which can only be written when they are combined with other vowels or consonants.
f) Bengali has its own punctuation marks such as '।'.

Bengali langauge has a huge number of characters. It is one of the richest languages of the world based on its resources. Table.1 shows samples of each type of Bengali characters in the next page. They are vowels, consonants, combined characters and special characters.

Vowel	অ আ ই ঈ উ ঊ
Consonant	ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড
Combined Character	শ্ব শ্বি শ্ম শ্মা স্ত্র স্বে স্থা স্ফু স্ম স্তী ষ্কা
Special Character	র্ষা র্ষী ষাঁ ক্যা ক্যে ক্যো ঝ্যো

Table. 1: Samples of characters

III. PROPERTIES OF DIGITAL BENGALI FONTS

It is particularly important to understand how computerized digital Bengali fonts work while developing an OCR. In Bengali ‘মা’ is a character that is a combination of ‘ম’ and the short form of vowel ‘আ’. This short form is represented by the short form itself with a circle concatenated with it. Here are few examples of them.

◌া ে◌ ◌ু ◌ী ি◌

These circles cause difficulty to make the character set of an OCR. It is therefore important for the developers to cover each and every pattern of a character that exists in Bengali language. A certain character ‘ম’ can have various patterns. These are shown below.

ম মা মু মি ম্যা মৌ মূ মি মী

Not only the characters can have such forms but also they may be combined to special characters which alone do not have any meaning. One of such special character is ‘Chandrabindu’. It remains on top of some characters such as ‘বাঁ’ to give them special pronunciation. Because of these very complex characteristics of digital Bengali fonts which are actually far different from real life’s handwritten font, it is very important to cope up with the differences of real life perception about fonts and the digital Bengali fonts that have their completely different ways of working.

IV. SYSTEM OVERVIEW

The system is based on Tesseract OCR Engine with the user interface developed in the Java Graphical User Interface platform [6]. The OCR Engine needs a library file called ‘traineddata’ to work on Bengali inputs. This file is consisted of some other files and a concatenation of those files. When the library file is put into a specific location, Tesseract can access it for its scanning and text generating purpose. Tesseract has a level of accuracy in its engine which is standard. This engine can work to its full potential provided the library file is accurate and rich. In our system, we have implemented the library file or traineddata in a very detailed manner. We have covered all sorts of Bengali letters for the ‘Solaimanlipi’ font. Bengali, as a language is quite complicated and it has got various types of letters including vowel, consonants, combined characters and other special types. We have developed a huge character set. It is important to mention that we need to uniquely identify each and every character in our system so that if the input file contains the character, the OCR recognizes it.

It is important to make all letters that are covered in the newspapers, books, journals, etc. We have prepared ‘tif’ formatted image files of them. Then those images are used to make the training files of the traineddata. Because of the complexity of the character set, the OCR may not always detect a character correctly even if the character is included in training files. Tesseract can be manipulated in both Windows and Linux. As Windows is relatively popular operating system, we have preferred to work on it and have installed the executable file of Tesseract in windows. In the command line of windows, we have executed several commands to prepare our training files from the images. Then those files are concatenated using commands to make the traineddata.

We have used NetBeans IDE to make the user interface. The entire system is formed as a zip package. The input can be selected using the interface. Output will be saved in the project’s folder. Then the user can use the OCR through the interface.

The accuracy of the output will vary according to the quality of input images. If the input file is a photo taken by a camera, it is important to notice if there is any shadow. If there is shadow, it will cause trouble for the OCR to differentiate between the shadow and the text. If the input file is obtained by scanning a document, it is likely to produce better quality outputs. The resolution of the scanned files should not be less than 400 dpi. If the input file is an electronically converted image, for instance, a converted image from a text file, the output is likely to be very good.

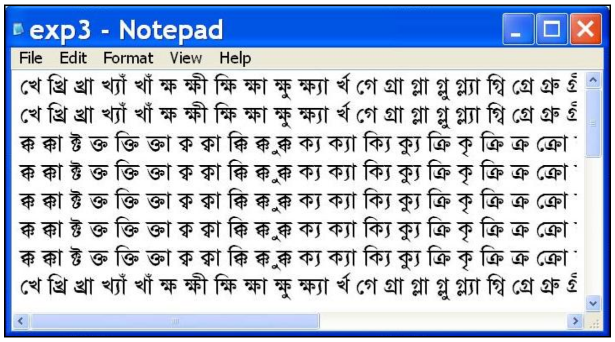
V. TRAINING PROCEDURE

A. Setting up the OCR Engine

We have installed Tesseract in Windows Operating System with the installer provided by Google. There is no user interface of Tesseract so we launched it by writing several commands in the command line of Windows.

B. Preparing text Files

Tesseract takes tiff formatted images files as inputs to make the training files. The image files can be a scanned image of a printed paper, an electronically converted image from a text file or an image taken by a camera. The usual practice is to convert text files to prepare images. We prepared eleven text files. It is important to note that the encoding of the text files needs to be UTF-8 otherwise the converter will not be able to read the Bengali text [9]. The first six text files contain the Bengali vowels, consonants, combined characters and unclassified characters. The last five text files contain paragraphs. In the first six files, each character is separated by a space. Each text file contains 1000 to 1700 characters. We had a long research and then gathered all possible patterns of characters in Bengali language and then repeated them for 5 to 10 times to fulfill the training requirement. Fig. 1 shows a sample text file that we used.



```
খে খ্রি খ্রা খ্যাঁ খাঁ ক্ষ ক্ষী ক্ষি ক্ষা ক্ষু ক্ষ্যা র্খ গে গ্রা গ্লা গ্লু গ্ল্যা গ্বি গ্রে গ্রু
খে খ্রি খ্রা খ্যাঁ খাঁ ক্ষ ক্ষী ক্ষি ক্ষা ক্ষু ক্ষ্যা র্খ গে গ্রা গ্লা গ্লু গ্ল্যা গ্বি গ্রে গ্রু
ক্ক ক্কা ক্ট ক্ত ক্তি ক্তা ক্ব ক্বা ক্কি ক্কু ক্ক ক্য ক্যা ক্যি ক্যু ক্রি কৃ ক্রি ক্র ক্রো
ক্ক ক্কা ক্ট ক্ত ক্তি ক্তা ক্ব ক্বা ক্কি ক্কু ক্ক ক্য ক্যা ক্যি ক্যু ক্রি কৃ ক্রি ক্র ক্রো
ক্ক ক্কা ক্ট ক্ত ক্তি ক্তা ক্ব ক্বা ক্কি ক্কু ক্ক ক্য ক্যা ক্যি ক্যু ক্রি কৃ ক্রি ক্র ক্রো
ক্ক ক্কা ক্ট ক্ত ক্তি ক্তা ক্ব ক্বা ক্কি ক্কু ক্ক ক্য ক্যা ক্যি ক্যু ক্রি কৃ ক্রি ক্র ক্রো
ক্ক ক্কা ক্ট ক্ত ক্তি ক্তা ক্ব ক্বা ক্কি ক্কু ক্ক ক্য ক্যা ক্যি ক্যু ক্রি কৃ ক্রি ক্র ক্রো
খে খ্রি খ্রা খ্যাঁ খাঁ ক্ষ ক্ষী ক্ষি ক্ষা ক্ষু ক্ষ্যা র্খ গে গ্রা গ্লা গ্লু গ্ল্যা গ্বি গ্রে গ্রু
```

Fig. 1: Sample text file for training

C. Making Image Files with Noise Margin

We have converted text files to images through a converter. The software is ‘jTessBoxEditor and it is used for training Tesseract. This software is based on java and it requires the java runtime environment (JRE) [8].

We converted the text files to tiff images using the editor. It is good to add some artificial noise to the training images which helps the OCR to work better on scanned image files. Noise refers to the uneven brightness in different parts of an image [5]. We set the noise margin’s value as 5 in jTessBoxEditor. The Editor generated a text file called box file for each tiff image that contains the characters of the images. Each character has a certain abstract edge and the edge is called a box. Each box has top, bottom, left, right coordinate value. For instance a particular box will have coordinates like the following example

ক 100 200 350 400

D. Editing Box Files

In Bengali fonts, short forms of vowels such as া ু ি ী are treated as separate characters. So, jTessBoxEditor which has Tesseract inside it split the short form of vowels. So, we needed to merge them to make characters like মা (ম+া) হা (হ+আ) সু (স+ু), শি (শ+ি). We merged them to get the requisite formatted characters such as মা, বা, চা. In total, we merged around 10,000 characters manually. Fig. 2 shows the merging process.

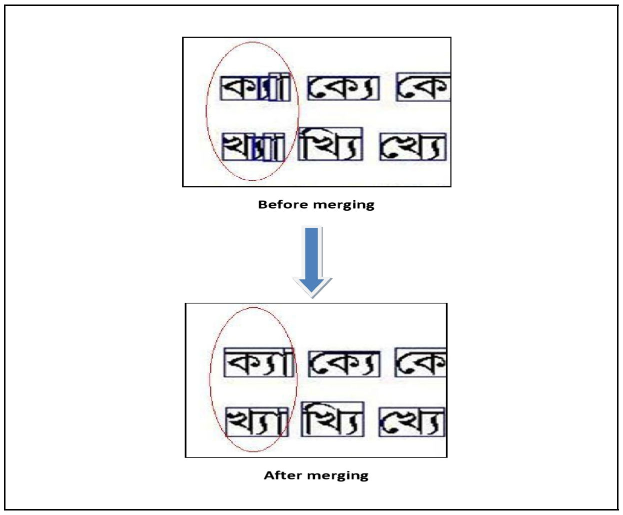
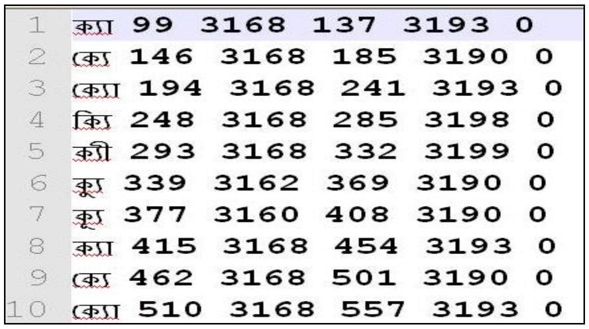


Fig. 2: Merging process

Fig. 3 shows a merged box file with coordinates.



```
ক্যা 99 3168 137 3193 0
ক্যে 146 3168 185 3190 0
ক্যো 194 3168 241 3193 0
ক্যি 248 3168 285 3198 0
ক্যী 293 3168 332 3199 0
ক্যু 339 3162 369 3190 0
ক্যূ 377 3160 408 3190 0
ক্যা 415 3168 454 3193 0
ক্যে 462 3168 501 3190 0
ক্যো 510 3168 557 3193 0
```

Fig. 3: Merged box file with coordinates

E. Running Tesseract for Training

For each pair of tiff image and box file, Tesseract is run in the training mode. It generates a file with file extension ‘tr’ for each pair of tiff image and box file. These files contain the training information.

F. Generating the Unicharset File

Tesseract needs to know the set of possible characters it can generate as output. For this reason, it needs a file named unicharset. To generate the file, we used the unicharset command provided in Tesseract’s manual.

G. Setting Font Properties

The latest version of Tesseract needs us to set the font’s properties for our language. Font properties indicate to the different properties a particular font can assume. For instance, an English font can have the properties of bold and italic. However, since Bengali can only be bold, it has only one property. In Tesseract, it is set by a text file that contains a line for each font. Since we worked on a single font, our file has only one line. The line is as follows:

solaimanlipi 0 1 0 0 0

This line refers to the following line

<fontname> <italic> <bold> <fixed> <serif> <fraktur>

We made the position of bold to 1 to make it set while we reset everything else. The font properties file is a UTF-8 encoded text file.

H. Clustering

When the alphabetical features of all the training images have been extracted, we need to cluster them to generate the prototypes. The character shape features can be clustered using the shapeclustering, mftraining and cntraining programs that are part of the OCR engine. We have used several commands to run those programs. We created three files named inttemp, pffmtable and normproto. The normproto data file contains the character normalization sensitivity prototype.

I. Making Optional Dictionary Data

We can use up to eight dictionary data files to train Tesseract but only two of them are relevant to our project. These files are optional and assist Tesseract to decide the probability of occurrence of different possible character combinations. They are the wordlist and the frequent wordlist. Our wordlist file contains around 2000 general random Bengali words and the frequent wordlist file contains around 600 very common Bengali words.

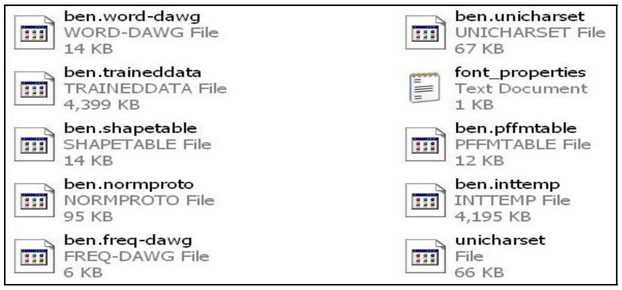
We have prepared a text file for each type of list in UTF-8 encoded form. In each file, a word is written in every line. It looks like the following lines:

আকাশ
বাতাস
মাটি

Then we have created two dawg (Directed Acyclic Word Graph) files using commands.

J. Generating Traineddata

After all these files have been prepared, we can run the final command to concatenate them to generate our traineddata[7]. The traineddata is now our library for working with Bengali letters. Tesseract's OCR engine uses its algorithm on this traineddata to produce Bengali outputs on inputs. The traineddata contains 18110 characters and 2617 words. Fig. 4 shows different files that have been generated in various steps and the concatenated traineddata.



ben.word-dawg
WORD-DAWG File
14 KB

ben.unicharset
UNICHARSET File
67 KB

ben.traineddata
TRAINEDDATA File
4,399 KB

font_properties
Text Document
1 KB

ben.shapetable
SHAPETABLE File
14 KB

ben.pffmtable
PFFMTABLE File
12 KB

ben.normproto
NORMPROTO File
95 KB

ben.inttemp
INTTEMP File
4,195 KB

ben.freq-dawg
FREQ-DAWG File
6 KB

unicharset
File
66 KB

Fig. 4: Training files and the traineddata

We need to add the 'ben' prefix to each filename as a requisite as 'ben' is the selected prefix for Bengali by International Organization for Standardization (ISO) for training Tesseract.

k. Managing Versions

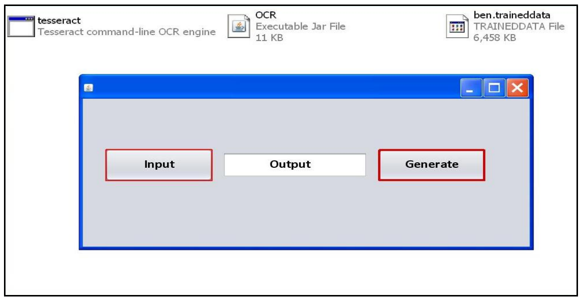
Version 3.03 of Tesseract OCR is used for generating outputs from Bengali text files. It is a beta version so it cannot be used for training the OCR engine and we have used the version 3.02 instead for making the library or traineddata for the OCR. Version 3.03 has been integrated to the system and has given better performance than that of Version 3.02. Though the entire training procedure is based on version 3.02, the integration has been good enough to produce the desired outputs.

l. Developing Graphical User Interface

We developed a graphical user interface as there is no user interface in Tesseract. This interface is implemented in NetBeans IDE using the built-in exec method in java. [10]

m. Packaging

The executable user interface file and the OCR are needed to be packaged. For this purpose, we have created a folder and placed three files in the folder. These files are the OCR's scanner, traineddata and the user interface's executable file. After launching the user interface file, one can select the inputs and the process them via the system. The outputs will be automatically generated in the package's folder. The user interface allows the users to name the output files. Fig. 5 shows the package of the system.



tesseract
Tesseract command-line OCR engine

OCR
Executable Jar File
11 KB

ben.traineddata
TRAINEDDATA File
6,458 KB

Fig. 5: System package

VI. ALGORITHM OF TESSERACT

Tesseract has a sophisticated algorithm. It uses search and fetch policy for generating output. When a character is read from the input file, Tesseract searches for a matching character in the images that are trained to the traineddata file. If it finds a matching character, it gets the coordinates of the character in that image and uses that coordinates to find the required position in the corresponding box file. It fetches the character from the box file with those particular coordinates and writes it to the output file. If there is no matching character, it fetches the character that resembles most to the character of the input file. However, if the character set is too large, it may mismatch a character with another character and that's why it doesn't always generate fully accurate output.

VII. RESULT

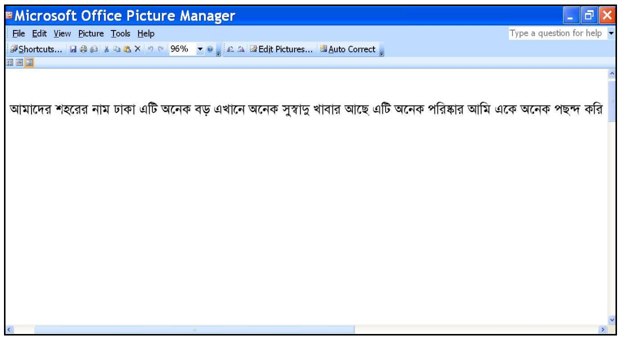
We have tested the OCR with two different sets of inputs. One set of inputs contains converted images from text and the other set contains scanned images of printed documents. In each set, there were 100 inputs and around 25 words and 70 characters in each input. The OCR does not detect the spaces among words in various cases. We have separated them manually to test the accuracy and calculated accuracy rate based on characters and words separately. Word accuracy checking policy is very strict. We have not considered any word as accurate which has a single mismatched character. We have calculated the accuracy of the OCR with the following equations.

Accuracy rate (Character) = (Number of correct characters/Number of total characters) × 100%

Accuracy rate (Word) = (Number of correct Words/Number of total Words) × 100%.

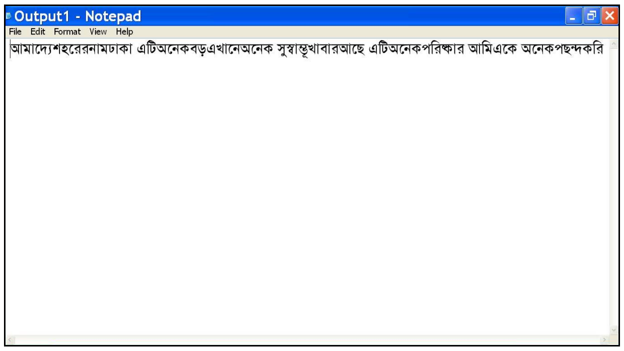
For scanned image files, the highest accuracy based on characters is 87.30% and the highest accuracy based on words is 40%. For image files that have been obtained by converting text files, the highest accuracy based on characters is 97.56% and the highest accuracy based on words is 90%. It is to be noted that there is significant difference between the percentage of accurate words for converted images and scanned images. This results from various factors. Firstly, a converted image is a perfect image with no skewness and perfect background. However, in case of scanned images, there may be skewness and the background may not be perfect. Moreover, scan quality also varies from machine to machine which has significant impact on the accuracy of an output. The OCR cannot handle it alone. The solution of this issue depends largely on the scanner's quality and images' resolutions. If the scanner used for capturing images is very good and the resolution is very high, the output will also be better. It is also a point to be noted that a single mismatched character excludes a word from the calculation of accurate words so word based accuracy level differs a lot from character based accuracy level for scanned images.

Fig. 6(a) shows a sample input and Fig. 6(b) shows a sample output.



আমাদের শহরের নাম ঢাকা এটি অনেক বড় এখানে অনেক সুস্বাদু খাবার আছে এটি অনেক পরিষ্কার আমি একে অনেক পছন্দ করি

Fig. 6(a): Sample input image for testing



আমাদ্যেশহরেরনামঢাকা এটিঅনেকবড়এখানেঅনেক সুস্বাস্তুখাবারআছে এটিঅনেকপরিষ্কার আমিএকে অনেকপছন্দকরি

Fig. 6(b): Output file of the sample input

Fig. 7(a), Fig. 7(b), Fig. 7(c) and Fig. 7(d) show the accuracies based on characters and words for converted and scanned images respectively.

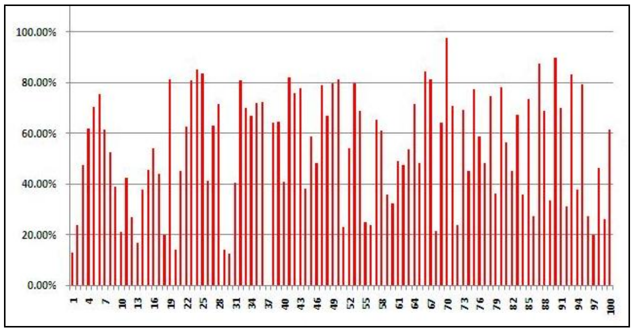


Fig. 7(a): Accuracy of converted images based on character

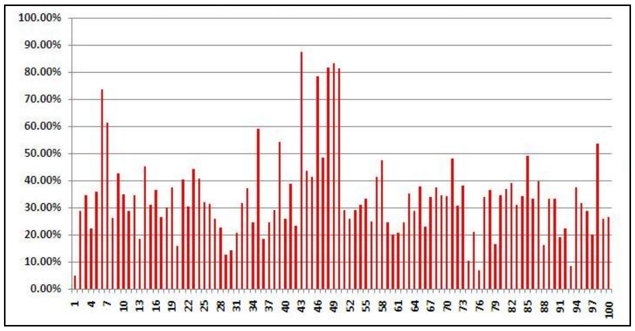


Fig. 7(b): Accuracy of scanned images based on character

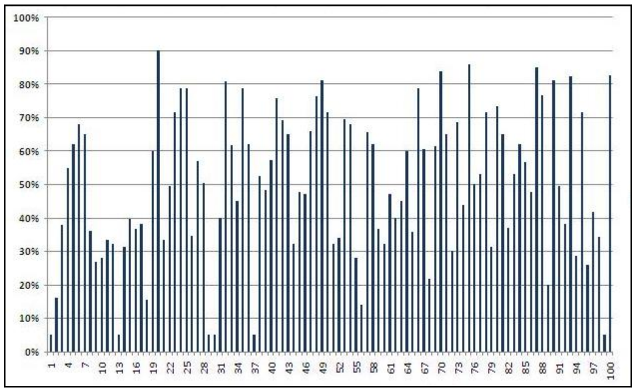


Fig. 7(c): Accuracy of converted images based on word

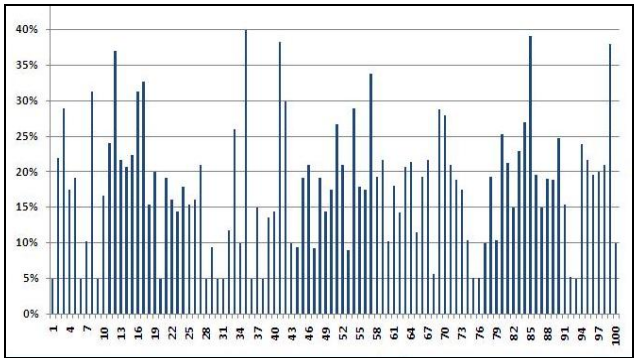


Fig. 7(d): Accuracy of scanned images based on word

VIII. CONCLUSION

The OCR is implemented on research purpose. It has got its limitations but it is an example of training an engine with expertise. We have gone through a lot of trial and error processes to find out which will be the best method to prepare images files from text and we have finally used the dedicated jTessBoxEditor. Lots of efforts have also been put in modifying the box files. Java GUI is used for the simplicity of the software. We have developed it as a desktop application so that it can be used offline. This project will be available for general purpose use for common users after adding further improvements such as integration of a spell checker with the OCR [2].

REFERENCES

[1] F. Y. Omee, S. S. Himel, M. A. N. Bikas,"A Complete Workflow for Development of Bangla OCR ", *International Journal of Computer Applications*, vol. 21, pp.2543-3483, May 2011.

[2] N. Uzzaman, M. Khan,"A Comprehensive Bangla Spelling Checker (2006)", ICCPB-2006.

[3] M. K. Shukla, T. Patnaik, S. Tiwari, S. K. Singh, Script Segmentation of Printed Devnagari and Bengali Languages Document Images OCR.

[4] U. Pal and B. B. Chaudhuri, "Identification of Matra Region and Overlapping Characters for OCR of Printed Bengali Scripts", *Intelligent Computing and Information Science Communications in Computer and Information Science*, vol. 135, pp. 606-612 2011.

[5] M.A Hasnat, S.M.M Habib, M. Khan, "A high performance domain specific OCR for Bengali script".

[6] M. A Hasnat, M. R Chowdhury, M. Khan,"Integrating Bengali script recognition support in Tesseract OCR.".

[7] Training Tesseract [Online] Available:https://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3.

[8] JTessBoxEditor Manual [Online]. Available: http://vietocr.sourceforge.net/training.html.

[9] Tesseract FAQs [Online] Available:https://code.google.com/p/tesseract-ocr/wiki/FAQ#Tesseract_FAQs.

[10] NetBeans GUI [Online] Available: https://netbeans.org/kb/docs/java/gui-functionality.html.