An open source Tesseract based Optical Character Recognizer for Bangla script

Md. Abul Hasnat   Muttakinur Rahman Chowdhury   Mumit Khan
Center for Research on Bangla Language Processing,
Department of Computer Science and Engineering,
BRAC University, 66 Mohakhali, Dhaka, Bangladesh.
mhasnat@gmail.com, sourogts@gmail.com, mumit@bracu.ac.bd

Abstract

BanglaOCR is currently the only open source optical character recognition (OCR) software for the Bangla (Bengali) script developed by the Center for Research on Bangla Language Processing (CRBLP). Tesseract, maintained by Google, is considered to be one of the most accurate free open source OCR engines currently available. In this paper, we present a new OCR for the Bangla/Bengali script that combines the recognition power of Tesseract and the Bangla script processing power of BanglaOCR by integrating the Tesseract recognition engine into BanglaOCR. We first present the complete methodology to build the combined OCR, followed by the implementation strategy. In this paper, we focus on the training data preparation process, Tesseract integration procedure and the post-processing techniques. The techniques described in this paper can be readily applied to build OCRs for other scripts as well.

1. Introduction

There have been a number of open-source research efforts since the mid 1980's aimed at building OCR systems for recognizing Bangla characters. Most of these efforts have focused on the recognition of individual characters, rather than on building a complete OCR system. It was not until 2006 when two separate OCR packages for the Bangla script – BOCRA and Apona-Pathak – were made available publicly [1]. The accuracy and other performance data however were never reported for either of these two packages. The Center for Research on Bangla Language Processing (CRBLP) released BanglaOCR – the first open source complete OCR software for Bangla – in 2007 [2]. BanglaOCR is a complete OCR framework, and has a recognition rate of up to 98% (in limited domains). It has its own pre-processor (which includes Otsu method for binarization, headline/matraa based segmentation, etc.); it uses DCT (Discrete Cosine Transform) as features; it has a Hidden Markov Model Toolkit (HTK) based recognition engine and has a dictionary based post-processor to correct spelling errors. In addition, BanglaOCR has a very flexible training method. Although the recognition rate of BanglaOCR is reported to be up to 98%, this rate is very sensitive to the typeface and the font size, as well as the document type. To maintain a reasonable recognition rate, a large amount of training data is required; the recognition accuracy reduces significantly when there are domain mismatches.

The Tesseract OCR engine was one of the top 3 engines in the 1995 UNLV Accuracy Test. Between 1995 and 2006 however, there was very little activity in Tesseract, until it was open-sourced by HP and UNLV in 2005; it was again re-released to the open-source community in August of 2006 by Google [3]. A complete overview of Tesseract OCR engine can be found in [4]. There are multiple efforts underway to integrate the recognition support of different scripts. The complete source code and different tools to prepare training data as well as testing performance is available at [5]. The algorithms used in the different stages of the Tesseract OCR engine are specific to the English alphabet, which makes it easier to support scripts that are structurally similar by simply training the character set of the new scripts. There are published guidelines about the procedure to integrate Bangla/Bengali language recognition using Tesseract OCR engine [6]. Detailed knowledge of the Bangla script as well as the character training procedure is also given in [6]. However, no real implementation is available till date.

"Tesseract based BanglaOCR" is an open source OCR software for Bangla script recognition that integrates Tesseract's excellent recognition engine into the rest BanglaOCR. It has a graphical user interface (GUI) that enables the user to load in a text image, recognize the image, and then use the built-in spelling

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.62

checker, which is capable of generating suggestions for misspelled words, to improve the accuracy. This software can be used as an important tool to build lexical resources for the Bangla language.

We briefly present our methodology in section 2, implementation details in section 3, then discuss the results in section 4, and then finally conclude.

2. Methodology

The subtasks behind building the complete OCR application are listed below:

1. Preparing Training data.
2. Preprocessing the document image.
3. Preparing Tesseract supported image.
4. Performing Recognition using the Tesseract engine.
5. Postprocessing the generated text output.

Among the sub tasks number 1 is independent than others. Tasks 2 to 4 are sequentially dependent on the success of the previous step.

2.1. Preparing Training data

A complete guideline to prepare training data for Bangla script is described in [6], which is what we followed to prepare the training data. Initially, we listed 340 (50 basic, 10 vowel modifiers and 270 compound characters) characters, and considered these as the basic units for training. With these units, we performed our experiment to estimate the requisite amount of training data. In the final training data set, we consider the following combinations:

- All vowels, consonants and numerals
- Consonants and vowel modifiers
- Consonants and consonant modifiers
- Combined consonants (compound character)
- Compound characters and vowel modifiers
- Compound characters and consonant modifiers

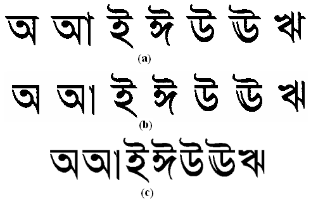
We performed a large number of experiments during training data preparation. The experiments were necessary to find the right combination of the training data that provides highest accuracy during recognition. We prepared fourteen different sets of training data with the following parameters:

- type of document image,
- image DPI information,
- font type & size,
- segmentation, and
- degradation.

The primary reason behind creating such large training data sets is Tesseract's limitation of supporting only 32 configuration files for each unit [5]. We automated the generation of the training images, which made the respective values of these parameters very important during the data preparation phase. The automated generation of the training data helped us to avoid the difficulties of collecting the large number of data units, such as the 3200 units described in [6], from real images. The parameter *type of document image* (TDI) encodes the source of the training data – either scanned or computer generated. *Computer generated images* (CGI) were the easiest way to generate training image from the specifications of the parameters: *image dpi* (IDPI), *font size* (FS) and *type* (FT). The intent of generating the scanned images (SI) is to make the training images more similar to the real images. To generate these images, we first prepared the paper documents following the specification of the typeface as well as the font size. Then, we scanned the paper documents with specific skew angles and also following the specification of the image resolution. We then de-skewed the images. The parameter *degradation* (DEG) was then applied to the prepared degraded images, which is applied on both CGI and SI. The application of *segmentation* (SEG) to the training images was one of the most important factors in improving the efficiency of the training data to a great extent. The reason behind applying this parameter is to make the training data units more similar to the characters/units in the test images that are provided to the recognizer. This approach of the preparing the training data significantly improves the accuracy rate. One important condition is that the segmentation algorithm must be same for both training and testing. An example of the importance of consistent segmentation is illustrated in Fig. 1 where (a) shows the training units without segmentation applied to these, (b) shows the training units after segmentation applied to these and (c) shows the condition of the test image before passing it on to the recognizer. Fig. 1 shows the significance of segmented training units, presumably because it exposes the underlying similarity between the training and testing units/characters. Also, note that applying segmentation reduces the amount of training data units. In Fig. 1(b), the unit আhas broken into অ and া, where া is already considered as a unit to be trained and hence it is no longer necessary to train the unit আ. Similar condition happened for many other units among the total 3200 units, which significantly reduced the amount of training data units. An example of segmented training units of consonant 'ক' is shown in Fig. 2. The top five combinations of the parameters are listed on Table 1. All of these combinations use font type of *SuttonyMJ*, font size of 24 and image DPI of 300. In our data preparation approach, we accumulated all character combinations in 13 images, numerals and modifier symbols in 7 images and only 1 image for degraded units.

Table 1: Top five combinations of the parameters to prepare training data

Name	Type of Document Image	Apply DEG	Apply SEG	Total Units
Set-1	SI	Yes	Yes	1955
Set-2	CGI	Yes	Yes	1963
Set-3 (Set-1+Set-2)	SI + CGI	Yes	Yes	3017
Set-4	SI + CGI	No	No	4236
Set-5	SI + CGI	No	Yes	4355



অ আ ই ঈ উ ঊ ঋ

(a)

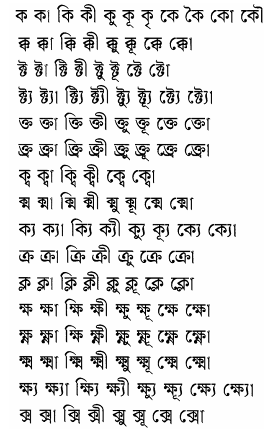
অ আ ই ঈ উ ঊ ঋ

(b)

অআইঈউঊঋ

(c)

Figure 1: Example of the effect of segmentation on training data



ক কা কি কী কু কূ কৃ কে কৈ কো কৌ
ক্ক ক্কা ক্কি ক্কী ক্কু ক্কূ ক্কে ক্কো
ক্ট ক্টা ক্টি ক্টী ক্টু ক্টূ ক্টে ক্টো
ক্ট্য ক্ট্যা ক্ট্যি ক্ট্যী ক্ট্যু ক্ট্যূ ক্ট্যে ক্ট্যো
ক্ত ক্তা ক্তি ক্তী ক্তু ক্তূ ক্তে ক্তো
ক্ত্র ক্ত্রা ক্ত্রি ক্ত্রী ক্ত্রু ক্ত্রূ ক্ত্রে ক্ত্রো
ক্ব ক্বা ক্বি ক্বী ক্বে ক্বো
ক্ম ক্মা ক্মি ক্মী ক্মু ক্মূ ক্মে ক্মো
ক্য ক্যা ক্যি ক্যী ক্যু ক্যূ ক্যে ক্যো
ক্র ক্রা ক্রি ক্রী ক্রু ক্রে ক্রো
ক্ল ক্লা ক্লি ক্লী ক্লু ক্লূ ক্লে ক্লো
ক্ষ ক্ষা ক্ষি ক্ষী ক্ষু ক্ষূ ক্ষে ক্ষো
ক্ষ্ণ ক্ষ্ণা ক্ষ্ণি ক্ষ্ণী ক্ষ্ণু ক্ষ্ণূ ক্ষ্ণে ক্ষ্ণো
ক্ষ্ম ক্ষ্মা ক্ষ্মি ক্ষ্মী ক্ষ্মু ক্ষ্মূ ক্ষ্মে ক্ষ্মো
ক্ষ্য ক্ষ্যা ক্ষ্যি ক্ষ্যী ক্ষ্যু ক্ষ্যূ ক্ষ্যে ক্ষ্যো
ক্স ক্সা ক্সি ক্সী ক্সু ক্সূ ক্সে ক্সো

Figure 2: Segmented training units for unit 'ক'

To complete the rest of tasks we followed the procedure in [6].

2.2. Preprocessing the document image

The main target of this step is to obtain information the about character/units after character segmentation is applied on test image. Another major concern of this step is to be able to read any format of input image. Hence the task of image acquisition as well as the extraction of the raw image data information was important. To perform the rest of the preprocessing subtasks except character segmentation we followed the techniques listed at [1]. A complete guideline about the requirements of segmented output is mentioned at [6] which we followed in our approach to build the segmenter. Writing a character segmenter for Bangla is a very comprehensive task which we experienced during the development of this application. At the end of this step we stored the information (position of left, right, top bottom) of each character/unit in an array.

2.3. Preparing Tesseract supported image

The goal of this stage is to generate an image with the Tesseract specific encoding. Tesseract is only capable to read an uncompress 1bit/8bit tiff format image. Using the segmentation information that we obtained from the previous stage, we generate an uncompressed 1bpp (bit per pixel) tiff image. We saved the image temporarily until the recognition output text is obtained.

2.4. Performing Recognition using Tesseract engine

The goal of this stage is to invoke Tesseract to recognize the temporary image and obtain the output text. There are three different ways (such as Tessnet dll file to use the API, source code and also executables) available to invoke Tesseract as a separate process to perform recognition and generate output text.

2.4. Post-processing the generated text output

In this stage we applied a two level post processing; where first level post processing is on the raw text obtained from the previous stage, and second level is applied on the first level preprocessor output to check the spelling mistakes and generate suggestions for the misspelled (out of dictionary) words.

The first level post processing has two subtasks where firstly it reorder the Unicode text and then correct known recognition mistakes following specific rules. The order/sequence of characters in Bangla text image is different from the sequence of Unicode text in most of the cases where the independent vowels are present with the consonants. The independent vowels ◌ে, ◌ৈ, ◌ো, ◌ৌ mostly suffers from this problem and hence reorder is obvious for them. In text image characters ◌ে and ◌ৈ appears before the consonant where in Unicode text they appear after the consonant. In the text image the single independent vowels ◌ো and ◌ৌ breaks down into two parts and hence it is necessary to identify the position of these vowels properly and then recombine them as they appear in the text. In the next subtask we correct the known recognition mistakes according to the following rules:

1. If two consecutive characters are vowel and both of them are same then delete any one.
2. If Onussar (◌ং), Zafala (◌্য), Aakar (◌া)and Daari (।) appears at the beginning of a word then remove it.
3. Change the position of Chandrabindu (◌ঁ) with dependent vowel using the following rule: Cons. + Chandrabindu + vowel_mod = Cons. + vowel_mod + Chandrabindu
4. Change the position of Zafala (◌্য) with its previous vowel modifier if any.
   Example: ক + ি◌ + ◌্য (কি্য) -> ক + ◌্য + ি◌ (ক্যি)
5. Replace Soreo (অ) followed by Aakar (◌া)with Soreaa (আ).
6. Replace Daari (।) with Aakar (◌া) if the Aakar (◌া) is following a space.
7. Replace Eekar (ে◌) followed by Aakar (◌া)with Ookar (ো◌).
8. Replace Eekar (ে◌) followed by part of Oukar (◌ৗ)with Oukar (ৌ◌).
9. If any lower modifier is followed by Aakar (◌া) and Eekar (ে◌) then the lower modifier will be eliminated.
10. If any exclamatory sign (!) appears in the middle of any word then replace it with Aakar (◌া).

The second level post processing invoke a dictionary consisting of 180K words. It performs spell checking operation on the text that we get from fist level task. To perform the task of this level, we followed the similar post processor proposed at [1]. However, we have to find out the rules for assigning same code. Initially the code rule table is build up by assigning same code to the characters which are visually similar. Later we increase the table entry by observing the frequency of the recognition incorrectness. An example of such group of characters which are assigned similar code is listed on Table 2.

Table 2: Example of the group of characters which are assigned similar code

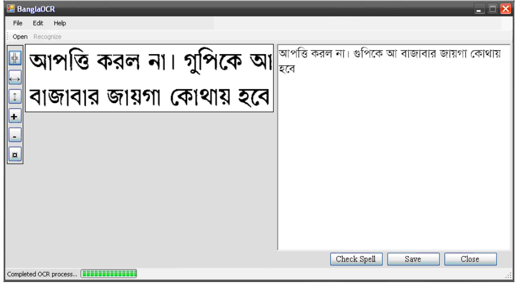
Groups	Characters
1	য়, ষ, য, ম, ঘ
2	খ, থ, স
3	গ, ণ, প, শ
4	চ, ঢ, ঢ়
5	ব, র

3. Implementation

The software is developed for both Windows and Linux environment. The windows version is completely developed using Visual C++ dot Net 2005. However, to develop the Linux version we followed several steps which are listed below:

1. Integrate preprocessing algorithms within Google Open source Project OCROpus (release 0.2) [7] as it has all necessary modules to invoke Tesseract as well as other image processing functions. Then create binary distribution of this.
2. Implement GUI in Java. The GUI has the capability to read image of any format, then invoke the OCROpus binary distribution to complete the recognition task and at last perform the post processing task using the necessary modules (written in Java) integrated with it.

A complete overview of the application is shown in Fig. 3 which show the condition of the OCR application after an image (shown in the picture box) is recognized. The left side of this GUI provides users the options to open an image, recognize an image and other preferences to enhance the image view in picture box. In the right hand side a text area will appear after recognition to show the output. At the bottom of the text area options are available to perform spell checking (2nd level post processing), save the text and close the output area.



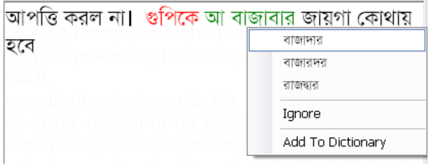
আপত্তি করল না। গুপিকে আ
বাজাবার জায়গা কোথায় হবে

আপত্তি করল না। গুপিকে আ বাজাবার জায়গা কোথায় হবে

Completed OCR process..

Figure 3: Complete overview of the application

The application can read image with any format. After the recognition of the image is performed, the picture box shows the temporary image which is generated from the segmentation information following the specification of Tesseract recognition engine. The "check spell" option on the application is developed following the functionality of the word processor application MSWord. After performing the spell checking it shows the output text maximum of three colors based on dictionary lookup. An example of such situation is presented in Fig. 4 where the text color with black indicates the correct words, the red colored text indicates an incorrect word without any suggestion and the green colored text indicates an incorrect word with suggestions. Users can right click over the incorrect words and find options to replace it with correct choice (for green color text only), ignore it and add the word into the dictionary.



আপত্তি করল না। গুপিকে আ বাজাবার জায়গা কোথায় হবে

Figure 4: Output text after performing spell check task

The application provides the opportunity to save the output text in several text file format depending on users choice. The output text is saved as Unicode text and hence enables further editing.

4. Result Analysis and discussion

We tested the application with a variety of different category images and observed the performance (Table 3). We measured the accuracy of the application considering character level recognition performance. The accuracy of the software mostly depends on the quality of the input image, and more specifically, on the image resolution. On average, we observed higher 93% accuracy for good quality images (resolution 300% or more). We observed low accuracy (below 80%) in the following document image types:

- screen-print images with small font size, and
- old document images with poor image quality.

Table 3: Accuracy analysis

Document Type	Num of images	Accuracy
Clean printed Document	30	93%
Printed book & newspaper	40	85%
Screen print image	15	70%

From the investigation of the poor performance, we identified that the quality of the image significantly affects the segmentation, leading to poor recognition performance. Screen print document image produces over segmentation of the characters while the other type image cause under segmentation. Analyzing the overall accuracy we set the preference of the resolution for the input image as 300 or more. We identified the problem in low-quality historical documents. The application and several test images are available at [8].

5. Conclusion

In this paper, we present complete procedure to build the Tesseract based BanglaOCR software. Methodology, implementation details as well as usage of this application are presented. Large amount of training data is prepared for this application and a massive experiment is performed with these training data to find out the right combination. While testing, an intermediate image is generated from the segmentation information following Tesseract specification. An efficient two level post processor is incorporated to enhance the performance and reliability of the software. The application is developed for both windows and linux environment. The performance evaluation of the application finds out the area of improvements which are listed as future works. The methodology outlined here would also be just as applicable to other members of the Brahmi family of scripts, such as Devanagri.

6. References

[1] Md. Abul Hasnat, S M Murtoza Habib and Mumit Khan. "A high performance domain specific OCR for Bangla script", Int. Joint Conf. on Computer, Information, and Systems Sciences, and Engineering (CISSE), 2007.

[2] http://sourceforge.net/project/showfiles.php?group_id=158301&package_id=215908. Last accessed: May 12, 2009.

[3] http://google-code-updates.blogspot.com/2006/08/announcing-Tesseract-ocr.html. Last accessed: May 12, 2009.

[4] Ray Smith, "An Overview of the Tesseract OCR Engine", Proc. of ICDAR 2007, Volume 2, Page(s):629 - 633, 2007.

[5] http://code.google.com/p/tesseract-ocr/. Last accessed: May 12, 2009.

[6] Md. Abul Hasnat, Muttakinur Rahman Chowdhury and Mumit Khan, "Integrating Bangla script recognition support in Tesseract OCR", Proc. of the Conference on Language and Technology 2009 (CLT09), Lahore, Pakistan, 2009.

[7] http://code.google.com/p/ocropus/. Last accessed: May 12, 2009.

[8] http://code.google.com/p/banglaocr/. Last accessed: May 12, 2009.