Odia Characters Recognition by Training Tesseract OCR Engine

Mamata Nayak
Department of CA
I.T.E.R., S'O'A University
Bhubaneswar, India

Ajit Kumar Nayak
Department of CS & IT
I.T.E.R., S'O'A University,
Bhubaneswar, India

ABSTRACT

Development of Optical Character Recognition (OCR) for an Indian script is an active area of research today. The presence of a large number of letters in the alphabet set, their sophisticated combinations and the complicated grapheme's they formed is a great challenge to an OCR designer. There are many application areas where, OCR can be used like, preserving old documents in electronics format, helping visually impaired persons to know the content of a document by transforming into speech, saving document images within limited space, making a electronic dictionary of words, preserving the ancient characters those are not included in the current set of characters of a language and many more. Currently, Tesseract, an open source OCR engine is considered as one of the most accurate FOSS OCR engines. Tesseract has already been designed to recognizing English, Italian, French, German, Spanish and Dutch and many more [11], as well as for few Indian languages such as Bengali, Tamil, Telugu, Malayalam. Similarly, Tesseract can be made to recognize other scripts if the engine can be trained with the requisite data. The objective of this work is to develop a training process for Tesseract OCR engine such that the engine will be capable of recognizing printed documents of Odia language used in the state of Odisha (formerly known as Orissa), India.

Keywords

Tesseract, OCR, UTF-8,UNLV, Odia.

1. INTRODUCTION (*Overview of the Tesseract OCR engine*)

Optical Character Recognition (OCR) system decreases the barrier of the keyboard interface between man and machine to a great extent. And it also help in office automation with huge saving of time and human effort. Overall, it performs automatic extraction of text from an image, exist in a variety of fonts, and be distorted in all sort of ways. Tesseract OCR, originally developed at Hewlett Packard from 1984 to 1994, is an open source (under Apache License 2.0) off-line optical character recognition engine[4,5]. Bristol, first started developing Tesseract as a PhD research project in HPLabs[6]. In the year 1995 it was sent to University of Navada, Les Vegas (UNLV), at there it proved its worth against the commercial engines of the time. In the year 2005 it was released by Hewlett Packard and University of Nevada, Las Vegas and presently it is partially funded and maintained by Google [7, 13]. However, output formatting, document layout analysis and graphical user interface is not support by the current version of it. We have used Tesseract version 3.01, which is released in Oct 2011, in the current work.

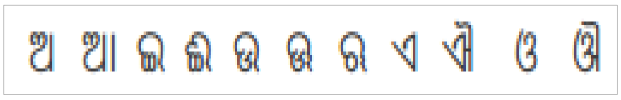
The Tesseract OCR engine was designed from the beginning to be language-independent, but the rest of the engine was developed for English. Its original design goal was that it should recognize white-on-black text. It follows a step-by-step pipeline procedure.

At the first stage, outlines of the text are gathered by nesting, into Blobs. These blobs are organized into text lines and are broken into words differently according to the kind of character spacing. After that the lines and regions are analyzed for fixed pitch or proportional text. Fixed pitch text is chopped immediately by character cells, however the proportional text is broken into words using definite spaces and fuzzy spaces. Then the recognition proceeds as a two-pass process. In the first pass, it attempt recognize each word in turn passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text. Thereafter the second pass is run over the page for the words those were not recognized well enough in the first pass. At last a final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate small cap text. We follow this architecture of Tesseract OCR engine to recognize the Odia characters.

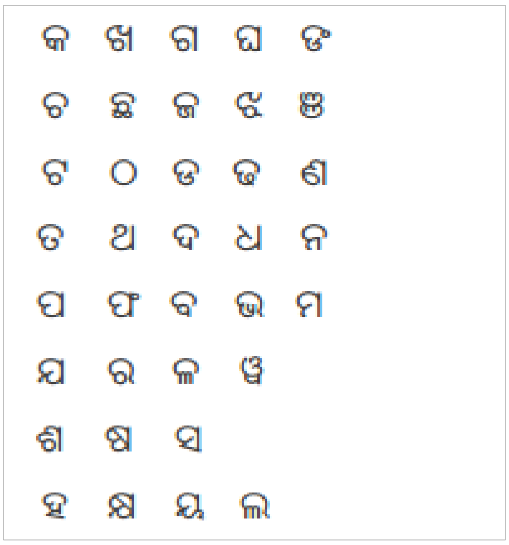
This is the first report attempt to train Tesseract for Odia script recognition, as of our knowledge. Content of this paper is organized as follows: In section 2, the properties of Odia script is presented, section 3 presents the training procedure in detail, followed by analysis of the result in section 4 and then conclusion in section 5.

2. Properties of Oriya Script

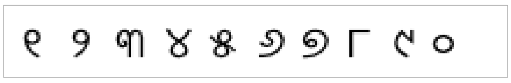
In the Odia script, there are 11 independent vowels shown in Fig. 1(a), 36 consonant characters shown in Fig. 1(b) and 10 digits shown in Fig. 1(c), these are called as basic characters.



(a) Vowels



(b) Consonants

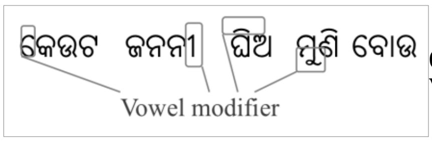


(c) Digits

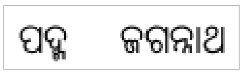
Fig. 1. Basic Characters of the Odia Script

Properties of the Odia script that are useful for building the recognition system [9, 10], briefly describe here. The Odia script is derived through various transformations from the ancient Brahmi script. The basic character names are identical to the names for corresponding characters in other scripts like Devanagari and Bengali. As like other Indian scripts also in Odia language, the concept of upper/lower case is absent. Among all these 11 independent vowels, 10 vowels have dependent forms (i.e. excluding first vowel). If the first character of any word is a vowel, then it persists in its independent form as shown in Fig. 2(a). Generally, a vowel takes a dependent form whenever it followed by a consonant, and the dependent form of the vowel is called as vowel modifier or allographs. Those are placed at the left or right or bottom or top or a combination, of the consonant as shown in Fig. 2(b). Similarly, a consonant modifier is placed when a consonant preceded or followed by another consonant as shown in Fig. 2(c).

(a) Independent form of Vowels



(b) Dependent form of Vowels



(c) Consonant modifier

Fig. 2. Vowel and Consonant modifiers

A variable-width encoding, UCS Transformation Format—8-bit (UTF-8), is used to represent every character in the Unicode character set. A Latin glyph in Unicode is represented as one character where as one glyph of Odia character set may contain string of Unicode characters, as shown in Fig. 3.

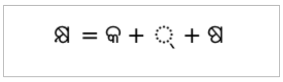


Fig. 3. String of Unicode characters for a single character

3. TRAINING PROCEDURE

A process used to Training Tesseract for recognizing a new language is known as training[1]. The training process is consists of several steps and each step gives us many information about how the language works. To prepare the training data a guideline is nicely written in [7], by following which we prepare our training data. For a language eight numbers of data files are required in tessdata sub directory in the training process[8]. These files are given as follows:

tessdata/xxx.inttemp

tessdata/xxx.normproto

tessdata/xxx.pffmtable

tessdata/xxx.unicharset

tessdata/xxx.freq-dawg

tessdata/xxx.word-dawg

tessdata/xxx.user-words

tessdata/xxx.DangAmbigs

where xxx refers to the three letter text, representing the language codes by following the ISO 639-3 standard[11].

As we pass through the steps of training process to train Odia language, each step is explain below. But while training we should make a trade-off between the speed of Tesseract's recognition versus its accuracy.

3.1 Prepare training data image

For the current experiment to collect the dataset it needs to determine the full character set to be used. At first prepare a text or word processor file containing all characters [1,3]. We consider a minimum of five numbers of samples for each character. Then to get the training data image, print the sample document out and scan it with 300DPI TIF B/W image, save the scanned file with .tif extension few lines of the image file is shown in Fig. 4.

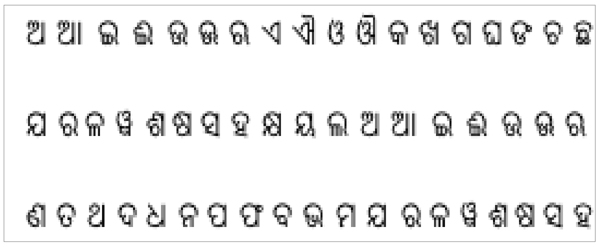


Fig. 4. Few lines of scanned training data image (odia.exp0.tif.)

Also it needs to save the training text as a UTF-8 text file for use in the next step, at where we have to insert the codes into another file.

3.2 Prepare box file

We need to prepare the box file for the above training image to generate the training files, using the following command:

```
$tesseract <image file> <box file> batch.nochop makebox
```

Example:

$tesseract <odia.exp0.tif> <odia.exp0> batch.nochop makebox

It generates a file named as odia.exp0.box, and if the training image consists of multiple pages then the box files need to be generated for each page. The box file is a text file consists of 6 columns. It includes the UTF-8 codes characters in the training image, one character per line (i.e. 1st column), along with the coordinates of the bounding box around the image (i.e. next 4 columns) and page number of the character in the image file (i.e. 6th column) as shown in the Fig. 5.

a 385 3194 413 3231 0

m 438 3193 477 3230 0

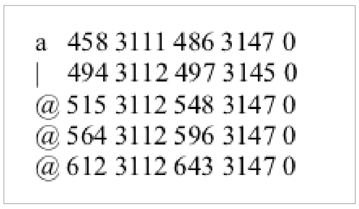
a 507 3191 540 3228 0

a 568 3190 601 3227 0

a 629 3189 661 3225 0

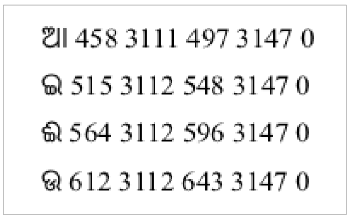
Fig. 5. Few lines of the box file generated for the image in Fig. 4.

The generated character set naturally have different to its current training, as shown in the Fig. 5. Because the box file is generated by considering the nature of English character so it fails to correctly generate box information for Odia character. Editing of the box file needs an editor that understands UTF-8 code to make the process lot easier. HTML, Notepad++, gedit are the editors that understands UTF-8. We use the editor gedit to edit the box file. If any particular character of the trained image is broken into two lines in the box file then it needs to manually merge the lines. As an example, few lines of the unedited box file is shown in Fig. 6(a) and as we can see the first two lines need to be combined to represent a single character as shown in Fig.6(b).



```
a  458 3111 486 3147 0
|  494 3112 497 3145 0
@ 515 3112 548 3147 0
@ 564 3112 596 3147 0
@ 612 3112 643 3147 0
```

(a) Before



```
ଆ 458 3111 497 3147 0
ଇ 515 3112 548 3147 0
ଈ 564 3112 596 3147 0
ଉ 612 3112 643 3147 0
```

(b) After

Fig. 6(a)(b). Few lines of the box file before & after manual modification

3.3 This Training the box file

Run Tesseract in training mode, for each of the training image and boxfile pairs by using the following command:

$tesseract <image file> <box file> nobatch box.train

Example:

$tesseract odia.exp0.tif odia.exp0 nobatch box.train

This will generate two files named odia.exp0.tr which contains the features of each character of the training page and another text file odia.exp0.txt. The odia.exp0.tr file contains the information about every character in the box file as follows:

UnknownFont <utf8 code(s)> 2

mf <number of features>

x y length dir 0 0

... (there are a set of these determined by <number of features>above)

cn 1

ypos length x2ndmoment y2ndmoment

The mf features are polygon segments of the outline normalized to the 1st and 2nd moments.

x= x position [-0.5.0.5]

y = y position [-0.25, 0.75]

length is the length of the polygon segment [0,1.0]

dir is the direction of the segment [0,1.0]

The cn feature is to correct for the moment normalization to distinguish position and size (eg c vs C and , vs ').

It is not required to edit the content of the *odia.exp0.tr* file and also the *odia.exp0.txt* contain a single newline with no text.

3.4 Compute the character set file

A program called unicharset_extractor is used to create the character set file. The box files are given as input, and it is invoked using the following command:

$unicharset_extractor <box file>

Example: *$unicharset_extractor odia.exp0.box*

It generates a data file named as unicharset, few lines of this file is given in Fig. 7.

```
48
NULL 0 NULL 0
ଅ 1 0,255,0,255 NULL 1  # ଅ [b05 ]x
ଆ 1 0,255,0,255 NULL 2 # ଆ [b06 ]x
ଇ 1 0,255,0,255 NULL 3  # ଇ [b07 ]x
ଈ 1 0,255,0,255 NULL 4  # ଈ [b08 ]x
```

Fig. 7. Few lines of the unicharset file

.

Each line of this file corresponds to one character in UTF-8 format. The character is preceded by a hexadecimal number representing a binary mask encoding its properties.

3.5 Prepare font_properties file

This file provides the information about the font family and the character properties such as isalpha, isdigit, isupper, islower. Create a text file that contains name of the font family and these properties with the value 0 or 1 (default value is 0) as given below:

<font family name> <[0][1]> <[0][1]> <[0][1]> <[0][1]> <[0][1]>

Because Oriya characters do not have lowercase and uppercase variance, the content of the file is written as:

LohitOriya 1 0 0 0 0

and save the file with the name font_properties. To proceed for the remaining stages it is necessary to rename the odia.exp0.tr file as the first word of the font_properties file i.e. LohitOriya.tr.

3.6 Clustering

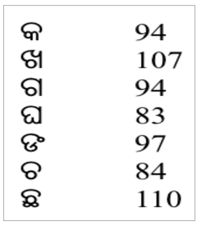
Clustering is necessary to create prototypes. This clustering is performed using the character shape features. We do this using two different programs called mftraining and cntraining. Then invoke the mftraining program using the following command:

$mftraining <filename.tr> -F font_properties -U unicharset

Example:

$mftraining LohitOriya.tr -F font_properties -U unicharset

The -F is used to include the font_properties file. And the -U option is used to include the unicharset file generated by unicharset_extractor. This will output two data files: pffmtable and inttemp. A third file is also created by this program, that will not be used further, called Microfeat . The file inttemp contain the shape prototype, but it cannot be open. The number of anticipated features for each character is shown as the content of the pffmtable file, a part of which is shown in Fig. 8.



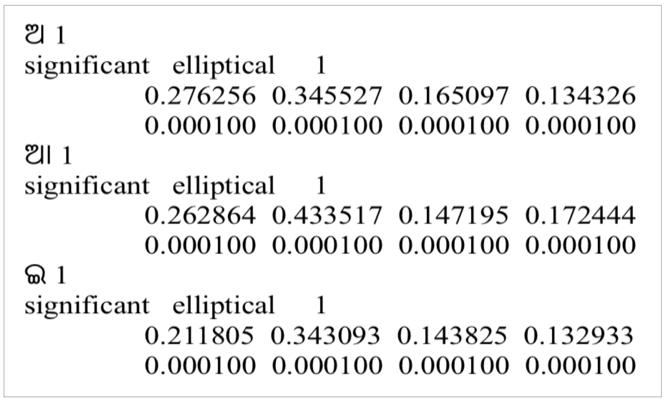
```
 କ    94
ଖ    107
ଗ    94
ଘ    83
ଙ    97
ଚ    84
ଛ    110
```

Fig. 8. Few lines of the pffmtable file

The following command is used to invoke the cntraining program: *$cntraining <filename.tr>*

Example: *$cntraining LohitOriya.tr*

This command produce the normproto data file, that perform the character normalization training for Tesseract, few lines of it is shown in Fig. 9.



```
ଅ 1
significant   elliptical    1
          0.276256  0.345527  0.165097  0.134326
          0.000100  0.000100  0.000100  0.000100
ଆ 1
significant   elliptical    1
          0.262864  0.433517  0.147195  0.172444
          0.000100  0.000100  0.000100  0.000100
ଇ 1
significant   elliptical    1
          0.211805  0.343093  0.143825  0.132933
          0.000100  0.000100  0.000100  0.000100
```

Fig. 9. Few lines of the pffmtable file

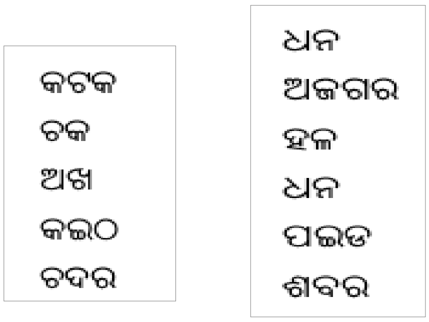
The following command will be used in case of multiple training data:

$mftraining -F font_properties -U unicharset <file1.tr> <file2.tr> ...

$cntraining <file1.tr> <file2.tr>

3.7 Prepare Dictionary

Tesseract usages few dictionary files for each language. We have used three files (frequent_word_list, words_list, user_char). Out of the three files, one file is a plain UTF-8 text file, and the other two files are coded as a Directed Acyclic Word Graph (DAWG). To make DAWG files first we need to create a word_list of Odia language that is formatted as a UTF-8 text file with one word per line. To get two UTF-8 text files, split the word list into two sets named as frequent_word_list and words_list. Few lines of these files shown in Fig. 10(a)&(b).



```
 କଟକ
ଚକ
ଅଖ
କଇଠ
ଚଦର
```

(a) frequent_words_list

```
ଧନ
ଅଜଗର
ହଳ
ଧନ
ପଇଡ
ଶବର
```

(b) words_list

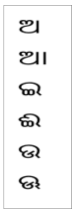
Fig. 10(a)(b). Few lines of UTF-8 dictonary files

These two files are converted into corresponding DAWG file using the command respectively as follows:

$wordlist2dawg <dictionaryfile> <dawg file>

Example: $wordlist2dawg words_list word-dawg

We put all the characters in the third dictionary file, called user-words as shown in Fig. 11.



```
ଅ
ଆ
ଇ
ଈ
ଉ
ଊ
```

Fig. 11. Few lines of UTF-8 dictionary files

3.8 The last step

By this time the training procedure is finished. The remaining work is to combine different generated files in different steps given in previous sections as follows. At first the required files are renamed by attaching a three letter prefix to each file name [12]. For Odia we used lang="ori". Thus, it is essential to rename the necessary files prefixed by lang+'.' provided within the two-column table (Table-1).

TABLE 1. Renaming of

Existing file names	**Modified file names**
unicharset	ori.unicharset
inttemp	ori.inttemp
pffmtable	ori.pffmtable
normproto	ori.normproto
freq-dawg	ori.freq-dawg
word-dawg	ori.word-dawg
user-word	ori.user-word

Now combine the files using the following command:

$combine_tessdata <3 letter language code>.

Example: *$combine_tessdata ori.*

It is necessary to give dot (.) at the end. This command results an output file lang.traineddata (i.e. ori.traineddata in this case) and then the generated combined file needs to be copied to the tessdata directory (usually: /usr/local/share/tessdata) . After this step the training process is finished and it is expected that Tesseract will be able to recognize any image file containing basic characters of odia script.

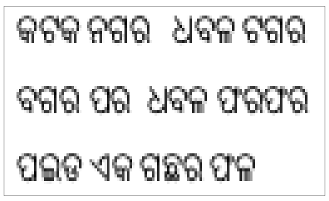
Now to test Tesseract, we need to give an input to Tesseract, an image file and a name for the file to be generated with the help of the following command:

$tesseract <image file> <text file> -l <lang>

Example:

$tesseract test.tif test -l ori

A part of the input image file and its transferred text file is as shown in the Fig. 12(a&b).



କଟକ ନଗର ଧବଳ ଟଗର
ବଗର ପର ଧବଳ ଫରଫର
ପଇଡ ଏକ ଗଛର ଫଳ

Fig. 12(a). Input image file test.tif

କଟକ ନଗର ଧବଳ ଟଗର

ବଗର ପର ଧବଳ ଫରଫର

Fig. 12(b). Generated test.txt file

4. Result Analysis

We determined the performance of Tesseract OCR for Odia language in recognizing the different Odia image documents. At first, we test a single page image file of isolated characters for the fonts having size 12pt, then increase the font size of the same page to 16pt, in both the cases we get 100% accuracy. Next we continue the testing of an Odia document image having multiple pages with mixed character sizes (10pt, 12pt and 16pt), we get 100% accuracy. However the font type of previously test documents are “Lohit Oriya”, so then we test for the font type “utkal” and find few errors. Summary of our testing results is shown in the table given bellow:

TABLE. 2 Result obtained

Image type	**Font Size**	**Accuracy**
Characters	Same (12pt)	100%
Characters	Increase (16pt)	100%
More than one page	Different size (10pt,12pt, 16pt)	100%
More than one page	Font type - utkal	98%

5. CONCLUSION

The biggest advantage of Tesseract OCR is its availability as open code. Thus anybody having the interest to study the working procedure, and skill to improve it can able to train it for a new language. In this paper, we present the step by step procedure to train Tesseract engine for Odia printed text document. At first we train the Tesseract for a particular font type of English language that has not been supported earlier by performing a series of test. We then train Tesseract to recognize the Odia character set, and observed the results. As we find editing the box file manually is a cumber-sum task (this language has a large character set), we try to generate the box file automatically [2]. Also we could be able to detect the vowels and consonants of Odia character set, however still we need to train Tesseract for dependent modifiers and other characters that exist in the Odia documents, in future.

6. ACKNOWLEDGMENTS

The authors would like to thanks Tom Nartker and John Burns for developing Tesseract as well as the ISRI group at UNLV and the google group for maintaining Tesseract OCR as open source.

7. REFERENCES

[1] Md.Abul Hasnat, Muttakinur Rsahman Chowdhury, Mumit Khan, "Integrating Bangla Script recognition support in Tesseract OCR", Proceeding of the Conference on Language & Technology, pp-108-112, 2009

[2] Nick White, " Training Tesseract for Ancient Greek OCR", October 2012 Available: www.eutypon.gr/eutypon/pdf/e2012-29/e29-a01.pdf

[3] Md.Abul Hasnat, Muttakinur Rsahman Chowdhury, Mumit Khan, "An open source Tesseract based Optical Character Recognizer for Bangla script", 10th Internal Conference on Document Analysis and Recognition (ICDAR), pp.671-679, ICDAR, 2009. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&atnumber=5277476

[4] Ray Smith, "An Overview of the Tesseract OCR Enigine", Proc. of ICDAR2007, Curitiba, Parana, Brazil, 2007. Available:

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?atnumber=4376991

[5] Jane Horgan , " Critical review of Tesseract", 4th April 2010 Available: https://muelli.cryptobitch.de/paper/2009-Tesseract-Review.pdf

[6] Sandip rakshit, Subhadip basu, "Development of a multi-user handwriting recognition system using tesseract open source engine", Proceeding International conference on C#IT, pp 240-247,2009.

[7] http://code.google.com/p/Tesseract-ocr/

[8] Md. Abdul Hasnat, " How to train Bangla and Devanagari script for tesseract engine", pp.1-4, 2008. Available: www.ias.ac.in/sadhana/Pdf2002Feb/pe990.pdf crblpocr.blogspot.com/.../how-to-train-bangla-devanagari.html

[9] B.B.Chaudhuri, U.Pal, M. Mitra, "Automatic Recognition of Printed Oriya Script", pp. 23-34, Sadhana, Vol.27, part 1,2002. Available:

[10] Sanghamitra Mohanty, Hemanta Kumar Behra, "A complete OCR Development System For Oriya Script", Proceeding of SIMPLE, Vol.4, 2004.

[11] Ray Smith, Daria Antonova, Dar-Shyang Lee, “Adapting the Tesseract Open Source OCR Engine for Multilingual OCR”, Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. Available: http://doi.acm.org/10/1145/1577802.1577804

[12] http://tesseract-ocr.googlecode.com/svn-history/r719/trunk/doc/tesseract.1.html

[13] http://en.wikipedia.org/wiki/List_of_ISO_639-2_codes

[14] George Nagy, “At the Frontiers of OCR”, Proceeding of the IEEE, Vol 80, No. 7, pp. 1093-1100