An Open-Source Tesseract Based Optical Character Recognizer for Bengali Language

Redwan Islam, Nahid Ferdaoush
Department of Computer Science and Engineering
Stamford University Bangladesh, 51 Siddheswari Rd, Dhaka
redwanislamdip@gmail.com, ferdaoush.nahid2014@gmail.com

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. OCR primarily works in two phases; they are character and word detection. In case of more sophisticated approach, an OCR also works on sentence detection to preserve documents' structures. In this paper, we would discuss the process of developing an OCR for Bengali language. Lots of efforts have been put on developing an OCR for Bengali. Though some OCRs have been developed, none of them is completely error free. For our thesis, we trained Tesseract OCR Engine to develop an OCR for Bengali language. Tesseract is currently the most accurate OCR engine. This engine was developed at HP labs and currently sponsored by Google. In Tesseract there are two option to training first one is Legacy Training and second is LSTM Training. We do both of them.

For Legacy Training We collect various Bengali article from various sources like journal, websites, E-newspaper etc. We prepared forty-three text files where we have used around 65000 character and 15000 words for it. But after doing all the work we drastically fail to produce a good trained-data. Because as we got 3% accuracy rate. Which is obviously not a good sign for a good OCR. So that's why we switch to Legacy Training to LSTM Training. Hopping for we can produce a good traineddata for our OCR. But after doing LSTM Training, when we want to check that how our trained-data perform. We found it perform very well. Which is very good sign for us. Because if you give any image (as input) and want to extract the image as text or want to convert PDF to DOC. Our trained-data will give you the output with 98% accuracy rate.

Introduction

1.1 Optical Character Recognition (OCR)
OCR stands for Optical Character Recognition. It is a widespread technology to recognize text inside images, such as scanned documents and photos. OCR technology is used to convert virtually any kind of image containing written text (typed, handwritten, or printed) into machine-readable text data. OCR is a part of character recognition:

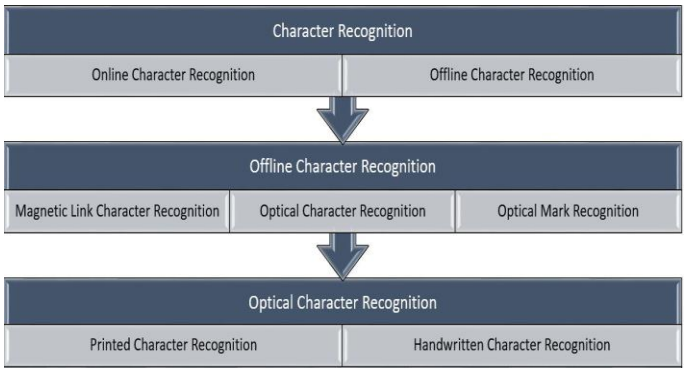

Figure 1.1: Types of Character Recognition

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 [2]. 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 are 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. 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 OCR and then the OCR will generate an editable text file for the user which is amendable. This file can be used to print or publish the required text. The software that performs the process is called Optical Character Reader or OCR.

1.1.1 Why OCR is used?

I. The most well-known use case for OCR is converting printed paper documents into machine-readable text documents for example (PDF to Word).
II. A machine-readable zone (MRZ) that can be Converted Image to Text. It's a very useable feature of OCR. For example, if you need image of a document to function like real text, where you can add new paragraphs, copy and paste, edit out an old reference, etc., OCR lets you do it.

1.2 Motivation

We see that many old books, important papers or documents get worn out or damaged with age. Nor, do we have any digital forms of these writings. There was no scanner to scan and store these documents. Thus, there is no other way to get the documents again other than typing the entire thing manually. Moreover, if we wanted to edit any part of the writing then we had to rewrite everything and correct the errors.

This can create overhead and unnecessary labor. Sometimes, even rewriting is not possible as the documents can get illegible. For this reason, we have thought to develop an OCR application to preserve such books and papers for our convenience. The application will be able to create editable text files so that we can edit or print the documents whenever we want. Furthermore, the GUI application will work quicker than typing or writing the whole thing manually.

Consequently, our primary goal is creating a GUI app so a non tech user can also use our app. putting multiple features in one platform like - Image to Text converter or PDF to Word converter. And make sure the accuracy rate is up to mark. Because we have already gone through some research paper on related works done previously. We found out that the accuracy level in various categories is not satisfactory. Therefore, we were motivated to increase the accuracy level. And lastly make sure that our app stays open source so that's why many people can use our app in daily basis.

1.3 Advantages and disadvantages of an OCR

Advantages	Disadvantages
OCR is much Cheaper than paying someone to manually enter large amounts of text.	OCR is not 100% accurate. It makes mistakes during the process.
Much faster than someone manually enter large amounts of text.	Sometimes OCR makes mistakes during processing, so we have to check the document all over carefully.

1.4 Scope for Bengali Language in OCR

There are plenty OCR available for different languages like English or Hindi. But for Bengali language there are few OCR available and most of them are not open source. If we want to use those OCR, then we have to pay some amount of money. But for a second, if we pay money to use their OCR. What is the guarantee that their OCR perform up to our expectation mark. Most of the OCR give accuracy rate 93% to 96% [1].

It is also not easy to find much information about developing an OCR for Bengali. Different projects have been done in different methods. Some developers used their own algorithms to develop OCR while some others used existing OCR engines to make OCR. It is not quite easy to develop an OCR for Indic languages like Bengali because of complexity. Bengali, for example, has diverse types of characters and they total 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 [2].

We preferred to work on an OCR engine for our thesis project. This engine called 'Tesseract' is well tested and it is the most accurate open-sourced OCR engine available. 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.

2. Literature Review

While progressing and planning for our thesis, we consulted many research papers regarding OCR. Since very less has been done on Bangla OCR, we studied the papers for Bengali, English and Hindi OCR and found fruitful information. Smith [3], discusses in his paper how OCR detects damaged characters easily. This was a big help for our research.

Hasnat [5], worked on HMM (Hidden Markov Model), a recognizer used for handwritten text and speech recognition, as there were very few attempts reported for printed character recognition. However, we decided to use Tesseract as character recognition, as it is the best possible open-source Optical Character Recognition engine.

Rakhsit [11], worked on recognizing handwritten text. They have used Tesseract 2.01 instead of building a new recognizer. They have used the handwritten version containing few Bengali characters. They used pen-based devices such as a stylus or tablet and gathered different handwritings from different people. Their first data set contained individual handwritten Bengali vowels, their second data set had Bangla consonant and third set contained digits. Their accuracy was around 90% for each set of data.

Bhowmick [4], on the other hand worked on Character recognition, she was able to create a trained-data using Tesseract Legacy Training Procedure for siyamRupali fonts. However, when we create trained-data for Kalpurush font. We found the accuracy rate is only 3%. Then we research little bit and found there is another procedure which is LSTM Training. Then we do the LSTM Training for Bengali language, and found 97.625% accuracy rate. We are the first one who do the LSTM Training for Bengali Language.

Though Bengali OCR is not a recent work, but there are very few mentionable works in this field. "BOCRA and Apona-Pathak" are two works which were made public in 2006. But they are not completely open-sourced. They offer some of their feature at free of cost. One of them is Unicode converter English to Bengali. When we check the converter, we found the accuracy is poor.

One professional OCR for Bengali has been developed by a company named 'Team Engine'. This project was financed by the government of Bangladesh. They demonstrated their OCR on web for a few months. However, currently the demonstration is not available. Its accuracy is said to be around 90%.

Another work was The Center for Research on Bengali Language Processing (CRBLP) released Bengali-OCR. The first open-source OCR software for Bengali published in 2007. Bengali-OCR is a complete OCR framework, and has a recognition rate of up to 98% but it also has many limitations in its domain.

3.1 Tesseract OCR Engine

Tesseract is an optical character recognition engine for various operating system. It is free software, released under the Apache License, version 2.0 and development has been sponsored by google since 2006.

The Tesseract OCR engine was originally developed as a primary software at Hewlett Packard between 1985 and 1995 and has been sponsored by google since 2006. Tesseract is the most accurate open-source OCR engine which uses Leptonica Image Processing Library for image processing purposes [12]. 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 language and UTF-8 characters, such as Bengali, several efforts have been made to modify its engine and the training system [3]. 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, we had to take adequate actions to make sure that Bengali 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 Bengali language. Additionally, it includes "Unicharset" to make multi-language handling easier. This function aids us we are going to be using four different Bengali 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 Bengali character compared to detect English characters. However, it can still detect Bengali character which is the main purpose

of our research.

Tesseract 3.03 added new training tool text2image to generate box/tiff files form text. It also has support for PDF output with searchable text. However, the text is only searchable, as it will be in PDF format cannot be edited. Tesseract latest stable version 4.00 is currently available Tesseract engine therefore, we used this engine in our research.

There are two ways to train tesseract for new language [10] first is Legacy Training and other is LSTM training. For our research purpose we tried both of them. And successfully create trained-data using the two ways. At the end we compared both trained-data. Which trained-data can give us more accurate result. Because at the end accuracy is the major factor.

3.2 Architecture

Since HP had independently-developed page layout analysis technology that was used in products, (and therefore not released for open-source) Tesseract never needed its own page layout analysis [7]. Tesseract therefore assumes that its input is binary image with optional polygonal text regions defined. Processing follows a traditional step-by-step pipeline, but some of the stages were unusual in their day, and possibly remain so even now.

The first step is a connected component analysis in which outlines of the components are stored. This was a computationally expensive design decision at the time, but had a significant advantage: by inspection of the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text. Tesseract was probably the first OCR engine able to handle white-on-black text so trivially. At first stage, outlines are gathered together, purely by nesting into blobs. Blobs are organized into text lines, and the lines regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells.

Proportional text is broken into words using define space and fuzzy spaces. Recognition then proceeds as two-pass process. In the first pass, an attempt is made to recognize each word in turn.

Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page.

Since the adaptive classifier may have learned something useful too late to make a contribution near the top of the page, a second pass is run over the page, in which words that were not recognized well enough are recognize again. A final phase resolves the fuzzy spaces, and checks alternative hypotheses for the X-height to locate small cap text.

3.3 Line and Word Finding

3.3.1 Line Finding - The line finding algorithm is one of the few parts of Tesseract that have previously been published [7]. The line finding algorithm is designed so that a skewed page can be recognized without having to de-skew, thus saving loss of image quality. The key parts of the process are blob filtering and line construction.

Assuming that page layout analysis has already provided text regions of a roughly uniform text size a simple percentile height filter removes drop-caps and vertically touching characters. The median height approximates the text size in the region, so it is safe to filter out blobs that are smaller than some fraction of the median height, being most likely punctuation, diacritical marks and noise. The filtered blobs are more likely to fit [7] a model of non-overlapping, parallel, but sloping lines. Sorting and processing the blobs by x-coordinates makes it possible to assign blobs to a unique text line, while tracking the slope across the page, with greatly reduced danger of assigning to an incorrect text line in the presence of skew.

Once the filtered blobs have been assigned to lines, a least median of squares fit is used to estimate the baselines, and the filtered-out blobs are fitted back into appropriate lines. The final step of the line creating process merges blobs that overlap by at least half horizontally, putting diacritical marks together with the correct base and correctly associating parts of some broken characters.

3.3.2 Baseline Fitting

Once the text lines have been found, the baselines are fitted more precisely a quadratic spline. This was another first for and OCR system, and enabled Tesseract to handle pages with curved baselines which are a common artifact in scanning, and not just at book bindings.

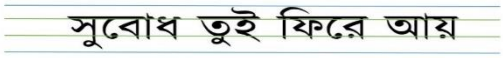

Figure 3.3.2: An example of fitted baseline

The baselines are fitted by partitioning the blobs into groups with reasonably continuous displacements for the original straight baseline [7]. A quadratic spline is fitted to the most populous partition, (assumed to be the baseline) by least squares fit. The quadratic spline has the advantage that discontinuities can arise when multiple spline segments are required. A more traditional cubic spline might work better.

Figure-3.3.2, it shows an example of a line of text with fitted baseline, descender line, mean-line and ascender-line. All these lines are “parallel” (the y separation is a constant over the entire length) and sometime it can be slightly curved if the text is curved.

3.3.3 Fixed Pitch Detection and Chopping

Tesseract tests the text lines to determine whether they are fixed pitch. Where it finds fixed pitch text, Tesseract chops the word into characters using the pitch, and disables the chopper and associate on these words for the word recognition step figure 2 shows a typical example of a fixed-pitch word.

Figure 3.3.3: A fixed pitch chopped word

3.3.4 Proportional Word Finding

Non fixed pitch or proportional text spacing is a highly non trivial task and tesseract solves most of these problems by measuring gaps in a limited vertical range between the baseline and mean line. Spaces that are close to the threshold at this stage are made fuzzy. So that a final decision can be made after word recognition.

3.4 Word Recognition

Part of the recognition process for any character recognition engine is to identify how a word should be segmented into character. The initial segmentation output from line finding is classified first. The rest of the word recognition step applies only to non-fixed pitch text.

3.4.1 Chopping Joined Characters

Tesseract attempts to improve the result by chopping the blob with worst confidence from the character classifier. Candidate chop points are found from concave vertices of a polygonal approximation [13] of the outline, and may have either another concave vertex opposite or a line segment. It may take up to 3 pairs of chop points to successfully separate joined character from the ASCII set. Chops are executed in priority order. Any chop that fails to improve the confidence the result is undone. But not completely discarded so that the chop can be re-used later by the associate if needed.

3.4.2 Associating Broken Characters

When the potential chops have been exhausted, if the word is still not good enough, it is given to the associate. The associate makes an A* (best first) search of the segmentation graph of possible combination of the maximally chopped blobs into candidate characters. The A* search proceeds by pulling candidate new states form a priority queue and evaluating them by classifying unclassified combination of fragments.

It may be argued that this fully-chop-then associate approach is at best inefficient, at worst liable to miss important chops, and that may well be the case. The advantage is that the chop-then-associate scheme simplifies the data structures that would be required to maintain the full segmentation graph. When the A* segmentation search was first implemented in about 1989, Tesseract's accuracy on broken characters was well ahead of commercial engines of the day.

3.5 Static Character Classifier

3.5.1 Features - An early version of Tesseract used topological features developed from the work of Shillman et. Al. [7] though nicely independent of font size, these features are not robust to the problems found in real- life images.

The breakthrough solution is the idea that the features in the unknown need not be the same as the features in the training data. During training, the segments of polygonal approximation are used for features, but in recognition, features of a small, fixed length are extracted from the outline and matched many-to-one against the clustered prototype features of the training data.

3.5.2 Classification - Classification proceeds as two-step process. In the first step, a class pruner creates a shortlist of character classes that the unknown might match. Each feature fetches, from a coarsely quantized 3-dimensional look up table, a bit-vector of classes that it might match, and the bit vectors are summed over all the features. The classes with the highest counts (after correction for expected number of features) become the short-list for the next step.

Each feature of the known looks up a bit vector of prototypes of the given class that it might match, and then the actual similarity between them is computed. Each prototype character class is represented by a logical sum of product expression with each term called a configuration, so the distance calculation process keeps a record of the total similarity evidence of each feature in each configuration, as well as of each prototype. The best combined distance, which is calculated from the summed feature and prototype evidences, is the best overall stored configuration of the class.

3.6 Linguistic Analysis

Tesseract contains relatively little linguistic analysis. Whenever the word recognition module is considering a new segmentation, the linguistic module (miss-named the permute) chooses the best available word string in each of the following categories: top frequent word, top dictionary word, top word, top upper-case word, top lower-case word (with optional initial upper), top classifier choice word. The final decision for a given segmentation is simply the word with the lowest total distance rating, where each of the above categories is multiplied by a different constant.

Words from different segmentation may have different numbers of character in them. It is hard to compare these words directly even where a classifier claims to be producing problematics, which Tesseract does not. This problem is solved in Tesseract by generating two numbers for each character classification. The first, called the confidence, is minus the normalized distance from the prototype.
This enables it to be a confidence in the sense that greater numbers are better, but still a distance. The second output called the rating, multiples the normalized distance from the prototype by the total outline length in the unknown character. Ratings for characters within a word can be summed meaningfully, since the total outline length for all characters within a word is always the same.

3.7 Adaptive classifier

It has been suggested [14][15] that OCR engines can benefit from the use of an adaptive classifier. Since the static classifier has to be good at generalizing to any kind of font, its ability to discriminate between different characters or between characters and non-characters is wakened. A more font-sensitive adaptive classifier that is trained by the output of the static classifier is therefore commonly [7] used to obtain greater discrimination within each document, where the number of fonts is limited.

Tesseract does not employ a template classifier, but uses the same features and classifier as the static classifier.

The only significant difference between the static classifier and the adaptive classifier, apart from the training data is that the adaptive classifier uses isotropic baseline/x-height normalization, whereas the static classifier normalizes characters by the centroid (first moment) for position and second moments for anisotropic size normalization.

The baseline/x-height normalization makes it easier to distinguish upper- and lower-case characters as well as improving immunity to noise specks. The main benefit of character moment normalization is removal of font aspect ratio and some degree of font stroke width. It also makes recognition of sub and superscripts simpler, but requires an additional classifier feature to distinguish some upper- and lower-case characters.

4.1 Properties of Bangla Font

Basic Bangla Scripts consists of total 11 vowels (Shoroborno), 39 consonant (Benjonborno) and 10 numerical. The basic characters are shown in table.

Vowels	অ, আ, ই, ঈ, উ, ঊ, ঋ, এ, ঐ, ও, ঔ
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, ব, শ, ষ, স, হ, ড়, ঢ়, য়, ৎ,
Numerical	০ ,১ ,২, ৩, ৪ ,৫, ৬, ৭, ৮, ৯, ১০

Table 4.1: Basic Character of Bengali Characters

4.2 properties in Bangla language

Some common properties in Bangla language are given below:

- Writing style of Bangla is from left to right.
- As English language there are no upper, and lower case in Bangla language.
- Most of the word, the vowels take modified shape called modifiers or allograph.
- There are approximately 253 compound characters composed of 2, 3 or 4 consonants.
- All Bangla alphabets and symbols have horizontal line at the upper part called "Matra" except এ, ঐ, ও, ঔ, খ, গ etc.

Table 4.2.1: Example of Bengali Modifiers

Vowel	Vowel Modifiers	Consonant	Consonant Modifiers
আ	া	ন	ন-ফলা
ই	ি	ম	ম-ফলা
ঈ	ী	র	র-ফলা

Table 4.2.1: Example of Bengali Modifiers

Compound Character	Formation of compound Characters
ন্ড	ন+ড
ক্ট	ক+ট

Table 4.2.2: Example Compound Characters

The "Matra" of a character remain connected with another character which also has "Matra" in a word. Those who have no "Matra" remain isolated in a word.

Bangla Characters contain three zones which are upper zone, middle zone and lower zone.

Figure 4.2: Dissection of Bengali Word

- Some characters including some modifiers, punctuation etc. have vertical stroke.
- Most of the characters of Bangla alphabet set have the property of intersection of two lines in different position. Many characters have the one or more corner or sharp angle property. Some characters carry isolated dot along with them.
- Figure 4.2, summarizes an example of Bangla character with zone and modifiers.

From all of these properties, there are approximately 310 types of patterns to be recognized for Bangla scripts [4].

- Some characters including some modifiers, punctuation etc. have vertical stroke.
- Most of the characters of Bangla alphabet set have the property of intersection of two lines in different position. Many characters have the one or more corner or sharp angle property. Some characters carry isolated dot along with them.
- Figure 4.2, summarizes an example of Bangla character with zone and modifiers.

From all of these properties, there are approximately 310 types of patterns to be recognized for Bangla scripts [4].

5.1 Legacy Training Introduction

Bengali as a language is quite complicated and it has various types of letters including vowels, consonant, combined character and other anonymous types. We developed a huge character set of all types after a long research. It is important to mention that we needed to uniquely identify each and every character in our system so that if the input file contains the character, the OCR recognizes it. We have covered all sort of Bengali letters for the Regular Kalpurush Font.

For Legacy training we installed Tesseract in windows operating system. Google provides an installed of Tesseract with built in English trained-data. We installed the installer. After installing the tesseract, a folder named "Tesseract-OCR" was created in the "Program Files (x86)" folder. There is no user interface for tesseract. So, we needed to use the command line to run it. We used the following commands to access the tesseract for training purpose.

For Legacy Training on windows platform:

```
cd c:\
cd Program Files (x86)
cd Tesseract-OCR
```

After the following commands we could access the Tesseract-OCR folder and its different components. Given below figure [5] shows a view of the Tesseract-OCR folder. Sometimes Tesseract install in Program files in C drive. So we have to make sure where we installed Tesseract. And then give the command according to the right path.

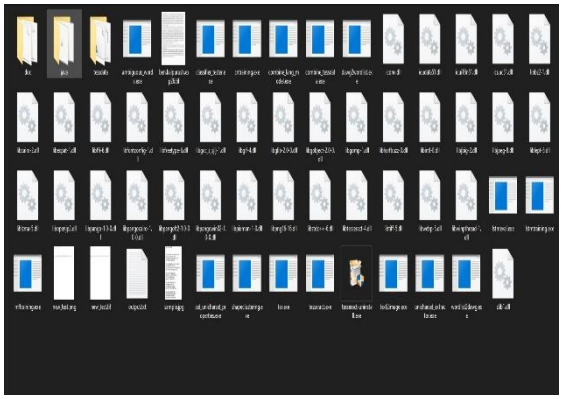

Figure 5.1: Tesseract Installation Folder

5.2 JTessBoxEditor

For legacy training we also need a software named JTessboxeditor. JTessboxeditor is a box editor and trained for Tesseract OCR, providing editing for box data of Tesseract formats. It can read common image formats, including multi page TIFF. The program requires Java Runtime Environment 7or later. JTessboxeditor is released and distributed under the apache license, V2.0 [4].

5.3 Working Procedure for Legacy Training

At first, we need a lots of character sets of “Regular Kalpurush Font” to prepare text files. Tesseract takes tiff format images files as inputs to make the trained-data. The usual practice is to convert text files to prepare images. 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 [2].

Figure 5.3: Block Diagram for Training Procedure

We rendered text to image and box files from the text files using JTessboxeditor. After that we manually edit those box files to prepare for training. And then we trained those files and create a trained-data. Bellow in figure [5], we described all the procedure in detail.

5.3.1 Preparing Text File

Figure 5.3.1: Demonstration of our Text File

We collect Bengali article from various sources like journals, E-newspaper, websites, etc. We prepared total 43 text files. And all those files contain full to bottom Bengali article. We assembled total 65000 characters and 15000 words for training. In figure 5.3.1, we demonstrate our text file.

5.3.2 Making Image Files with Noise Margin

We converted text files to image through a converter. There is a software named “Jtessboxeditor” for Tesseract. This software is based on java and it requires the java runtime environment (JRE). We converted the text files to tiff images using the editor. Past research suggest that it is good to add some artificial noise to the training images which helps the OCR to work better in scanned image files [4].

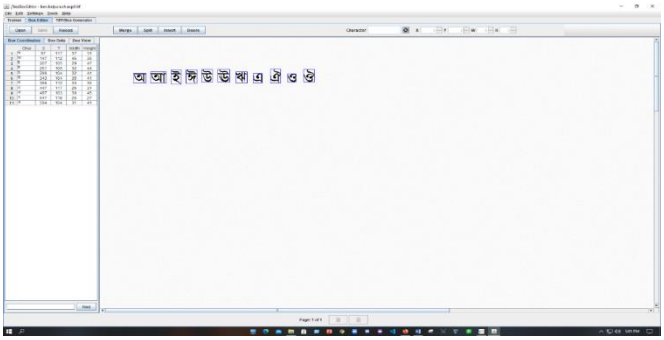

Figure 5.3.2: Conversation Text Files to Image Files

We set the noise margin’s value as 5 in Jtessboxeditor, which is the higher value. The editor generated a text file called box file for each tiff image. Box file is UTF-8 encoded test file that has the coordinates of all the characters in the training images along with the characters. Each character has a certain abstract edge and the edge is called a box. Each box will have coordinated like that following example.

5.3.3 Editing Box Files

In Bengali fonts, short forms of vowel া ী ি ু are treated as separate characters. So Jtessboxeditor which has Tesseract’s training engine inside it split the short form of the vowels. So we needed to merge to make character like ম (ম+া) হ (হ+আ).

The editor has the option. We used them to get the requisite formatted characters such as মা, নানা, চাচা.

However, for special characters like ◌ং ◌ঃ merging the box files need special measures, we needed to manually edit the images with photo editor and then generated box files and edit them.

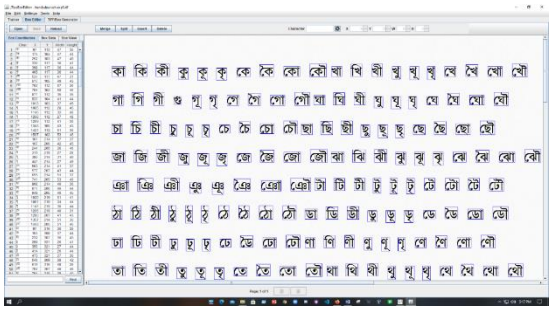

Figure 5.3.3: Merged Split Characters

5.3.4 Running Tesseract for Training

For each pair of tiff image and box file, we ran Tesseract for training. It generated a file with extension "tr" for each pair of tiff image and box file. These files contain the training information. Tesseract scans a character form the input, tries to match it with a shape available in the tiff image which is in the trained-data and write the matching character from the box file in the output.

For Legacy Training on windows platform:
tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num] box.train
tesseract ben.kalpurush.exp0.tif ben.kalpurush.exp0 box.train

So we needed to cover all sorts of available shapes in the training images. We used the following command for creating each "tr" file.

We have to run this command over and over. We have total forty-three file that's why we run this command over forty-three times.

5.3.5 Generating the Unicharset File

Next we must create a unicharset data file, which lets Tesseract know the set of possible characters it can output. For this we need to use the unicharset_extractor program on the box files generated above.

For Legacy Training on windows platform:
unicharset_extractor [lang].[fontname].exp0.box…[lang].[fontname].expN.box
unicharset_extractor ben.kalpurush.exp0.box……ben.kalpurush.exp43.box

However, unlike previous process, we do not need to type the same command over and over for every box files separated by a space.

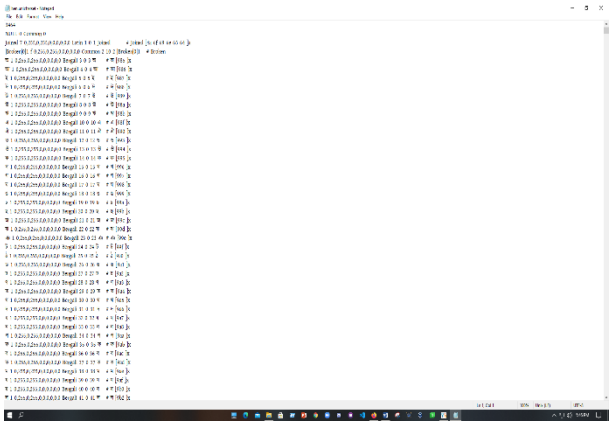

Figure 5.3.5: Unicharset Sample

5.3.6 Font Properties

Subsequently, we need to generate the font_properties files. 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.

E.g. kalpurush 0 0 0 0 0 means that the font name is kalpurush with no styling presents.

5.3.7 Shape Clustering

When the character features of all the training pages have been extracted, we need to cluster them to create the prototypes, the character shape features can be clustered using the shape clustering, mf training and cntraining programs:

For Legacy Training on windows platform:
Shapeclustering –F font_properties –U unicharset lang.fontname.exp0.tr……..
Lang.fontane.expN.tr
E.g. shapeclustering –F font_properties –U unicharset ben.kalpurush.exp0.tr……..
Ben.kalpurush.exp43.tr

Shapeclustering creates a master shape table by shape clustering and generates a shape table file. As we are training for Indic language such as Bengali, Tamil, Malayalam etc. We are going to use the shapeclustering method. This method is only used for Indic language.

For Legacy Training on windows platform:
mftraining –F font_properties –U unicharset –O lang.unicharset lang.fontname.exp-.tr ….. lang.fontname. exp43.tr
E.g. mftraining –F font properties –U unicharset –O ben.unicharset ben.kalpurush.exp0.tr … ben.kalpurush.exp43.tr

The –U file is the unicharset generated by unicharset_extractor above, and ben.unicharset is the output unicharset that will be given to combine_tessdata. mftraining will output to two other data files: inttemp and pffmtable. The inttemp file contains information about the shape prototypes, whereas, pffmtable contains the number of expected features for each character we are training.

For Legacy Training on windows platform:
cntraining lang.fontname.ecp0.tr …….. lang.fontname.expN.tr
E.g. cntraining ben.kalpurush.exp0.tr ….. ben.kalpurush.exp43.tr

This command will create the normpotro data file which is the character normalization sensitivity prototypes.

5.3.8 The Final Crunch of Legacy training

Finally, all of our data files are created and ready for use. Now it was time to create the trained-data. The trained-data file is simply the concatenation of all the data files we had created above. For this, we first sorted out the files and added the prefix "ben" to all of them.

Before	After
Shapetable	ben.Shapetable
normproto	ben.normproto
inttemp	ben.inttemp
pffmtable	ben.pffmtable
unicharset	ben.unicharset

Table 5.3.8: Renaming All the Files with Language Prefix

Now we were all set to generate the trained-data file. For this, we ran the command "combine_tessdata" On them as follows:

For Legacy Training on windows platform:
combine_tessdata lang.
E.g. combine_tessdata ben.

However, if in windows platform there is a software called "Serak Tesseract Trainer", using this software we can simply automate the process. Manual labor is greatly reduced if we used this software carefully. But when we using this software we do face problem in shapeclustering phase. So that why we choose not to use that.

5.4 Accuracy Rate

After all the work done when we check the result of our trained-data, we surprisingly got 97% error rate. Which is very shocking for us. Because if you give any input (as image) and want to extract the image as text. Our trained-data will give you the output with 3% correction rate. Which is obviously not a good sign for a good OCR. Below we provide some sample.

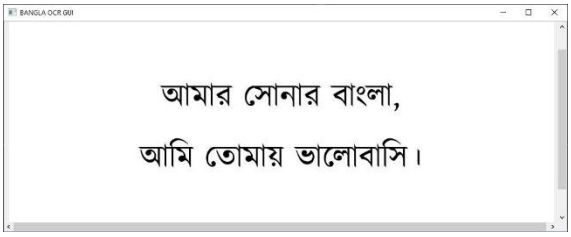

Figure 5.4.1: Random Image with Bengali Text

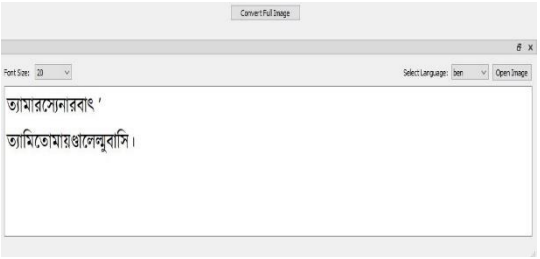

Figure 5.4.2: After extracting with Our Trained-data

As we can see, we give input a random image (as input) in our app to extract text form it. And after extracting we get text as an output. If we compare the input image and output text, it's completely not matching with each other.

There is no space. Letters correction is miserable. It cannot detect the modifier as well as the punctuation. As we said earlier with our Legacy trained-data we achieve only 3% accuracy rate. After doing all these training procedures we ensure that Legacy training is not suitable for Indic language. So that's why we switch Legacy to LSTM training.

6.1 LSTM Training Initiation

Tesseract 4.00 includes a new neural network-based recognition engine that delivers significantly higher accuracy (on document images) than the previous versions, in return for significant increase in required computer power [10]. On complex languages however, it actually be faster than base tesseract. And delivered more precise accuracy.

Neural networks require significantly more training data and train a lot slower than base Tesseract. For Latin based languages, the existing model data provided has been trained on about 400000 text lines spanning about 4500 fonts. For other scripts, not so many fonts are available, but they have still been trained on a similar number of text lines. Instead of taking a few minutes to a couple of hours to train, Tesseract 4.00 takes a few days to couple of weeks. For our case it took almost 2 weeks to get error rate 2.9 at 74000 iterations.

6.2 Neural Network

Neural network is a series of algorithm that endeavors to recognize underlying relationship in a set of data through a process that mimics the way the human brain operates [10]. In this sense, natural neural network refers to system to neurons, either organic or artificial in nature. Natural neural networks can adapt to changing input; so the network generate the best possible result without needing to redesign the output criteria.

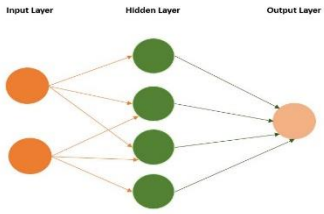

Figure 6.2: Neural Network

6.3 Neural Network System in Tesseract

Tesseract 4.00 includes a new neural network subsystem configured as a text line recognizer. It has its origins OCRopus' python based LSTM implementation, but has been totally redesigned for Tesseract in C++. The neural network system in Tesseract pre-dates Tensor-Flow, but is compatible with it, as there is a network description language called Variable Graph Specification Language (VGSL), that is also available for Tensor-Flow.

The idea of VGSL is that it is possible to build a neural network and train it without having to learn a lot of anything. There is no need to learn Python, Tensor-Flow, or even write any C++ code. It is merely required to understand the VGSL specification language well enough to build a syntactically correct network description. Some basic knowledge of what the various neural network layer types are and how they are combined will go a very long way.

6.4 Integration with Tesseract

The Tesseract 4.00 neural network subsystem is integrated into Tesseract as a line recognizer. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

The neural network engine is the default for 4.00 to recognize text from an image of a single use setpagesegemode (PSM_RAW_LINE). This can be used from the command line with –psm 13. This neural network engine has been integrated to enable the multi- language mode that worked with Tesseract 3.04, but this will be improved in a future release [10].

Vertical text is now supported for Chinese, Japanese and Korean, and should be detected automatically.

There is multiple option for training-

Figure 6.4: LSTM Training Multiple Options**.**

- Fine Tune: Starting with an existing trained language, train on your specific additional data. This may work for problems that are close to the existing trained-data, but different in some subtle way, like a particularly usual font. May work with even a small amount of training data.
- Cut off the top layer (or some arbitrary number of layers: From the network and retrain a new top layer using the new data. If fine tuning doesn't work, this is most likely the next best option. Cutting off the top layer could still work for training a completely new language or scripts, if you start with the most similar looking scripts.
- Retrain from scratch: This is daunting task, unless you have a representative and sufficiently large set for your problem. If not, you are likely end up with over-fitted network that does really on the training data, but not on the actual data. In our case we train from scratch.

While the above options may sound different, the training steps are actually almost identical, apart from the command line, so it is relatively easy to try it all ways, given the time or hardware to run them in parallel. For 4.00 at least the old recognition engine is still present. And can also be trained.

6.5 Requirements for Training

Hardware – Software Requirements

- There are some requirements you have to fulfill, if you want to train tesseract. At time of writing, the LSTM training only works on Linux. (Mac OS almost works; it requires minor hacks to the shell scripts to account for the older version of bash it provides and differences' in mktemp.
- Windows is unknown, but would need msys or cgwin. In our case we use Linux Ubuntu 18.04 version to train the tesseract.
- As for running Tesseract 4.0.0, it is useful, but not essential to have a multi-core (we found 8 core is good enough because if you use Linux in windows using virtual box than you can give 4 cores to Linux system and other 4 to the Windows OS) machine, Enough Ram is needed. In our training days we start training with 4GB Ram and found it's not enough. Training process are crashed whenever we running command for tesseract, because of short memory. Than we extended our memory 4 GB to 12 GB. After extended our memory we see training process are running smoothly. But if you training tesseract for above 100-250 pages you need a strong and powerful device with memory of 32 GB.

While the above options may sound different, the training steps are actually almost identical, apart from the command line, so it is relatively easy to try it all ways, given the time or hardware to run them in parallel.

6.6 Installing Tesseract Procedure in Linux

These are the instructions for installing Tesseract form git repository. We should be ready to face unexpected problems.

6.6.1 Installing with Auto-conf Tools

In order to do this, we must have auto-make, libtool, leptonica, make and pkg-config installed. In addition, you need Git and C++ compiler. On Debian or Ubuntu we can install all required packages like this.

```
sudo apt-get install automake ca-certificates G++ git libtool libleptonica-dev make pkg-config
```

The optional manpages are built with asciidoc

```
sudo apt-get install automake ca-certificates G++ git libtool libleptonica-dev make pkg-config
```

Now we want to build the Tesseract training tools as well, we will also require pango:

```
sudo apt-get install libpango1.0-dev
```

Afterwards, to clone the master branch on our computer we do this:

```
sudo git clone https://github.com/tesseract-ocr/tesseract.git

Sudo git clone –depth 1 https://github.com/tesseract-ocr/tesseract.git
```

Or to clone a different branch/version:

```
Sudo git clone https://github.com/tesseract-ocr/tesseract/git --branch <branch name>? –Single-branch
```

Tesseract require Leptonica v1.74 or newer. If your system has only older versions of Leptonica, you must compile it manually.

Finally run these:

```
cd tesseract
./autogen.sh
. /configure
make
sudo make install
sudo ldfconfig
```

The above does not build the tesseract training tools. We need the training tools. To download training tools, we need the following library.

```
Sudo apt-get install libicu-dev
Sudo apt-get install libpango1.0-dev
Sudo apt get install libcario2.dev
```

To build Tesseract with training tools, we have to run these command

```
cd tesseract
./autogen.sh
. /configure
make
sudo make install
sudo ldfconfig
make training
sudo make training-install
```

6.7 Post Install Instruction

There are two parts to install for Tesseract, the engine itself, and the traineddata for a language. The above installation commands install the Tesseract engine and training tools. They also install the config files e.g. those needed for output such as pdf, tsv, hocr, alto, or those for creating box files such as lstmbox, wordstrbox. In addition to these, traineddata for a language is needed to recognize the text in images.Three types of traineddata files for over 130 languages and over 35 scripts are available in tesseract-ocr GitHub repos.

```
([tessdata](https://github.com/tesseract-ocr/tessdata), [tessdata_best](https://github.com/tesseract-ocr/tessdata_best) and [tessdata_fast](https://github.com/tesseract-ocr/tessdata_fast))
```

When building from source on Linux, the tessdata configs will be installed in `/usr/local/share/tessdata` unless you used `./configure --prefix=/usr`. Once installation of tesseract is complete, don't forget to download the language traineddata files required by you and place them in this tessdata directory (`/usr/local/share/tessdata`).

If you want support for both the legacy --oem 0 and LSTM --oem 1 engine, download the traineddata files from [tessdata](https://github.com/tesseract-ocr/tessdata). Now Tesseract is ready to perform LSTM training.

6.8 Tesseract Training Procedure from Scratch

Overview of Training process

The overall training process is similar to Legacy Training. Conceptually the same:

- Prepare training text.
- Render text to image + box file. (Or create hand-made box files for existing image data.)
- Make unicharset file. (Can be partially specified, i.e. created manually).
- Make a starter trained-data from the unicharset and optional dictionary data.
- Run tesseract to process image + box file to make training data set.
- Run training on training data set.
- Combine data files.

The key differences are:

- The boxes only need to be at the t-extline level. It is thus far easier to make training data from existing image data.
- The .tr files are replaced by. lstmf data files.
- Fonts can and should be mixed freely instead of being separate.
- The clustering steps (mftraining, cntraining, shapeclustering) are replaced with a single slow lstmtraining step.

6.9 Prepare Training Text

For LSTM training we create a text files called "ben.training_text". And we stored their articles. We collect these articles in various sources like books, journals, E-newspaper, website etc.

6.10 CORPUS

Now we need to build a corpus, basically corpus is like a storage folder where we have to store everything that we need to create a trained-data. Below figure [15], we demonstrate the corpus.

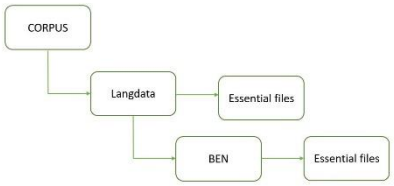

Figure 6.10.1: CORPUS

Basically our corpus is divided into two section. Our corpus folder name is Langdata. In Langdata there is some essential file –

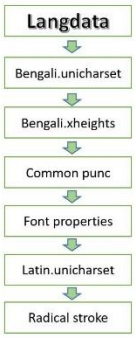

Figure 6.10.2: Demonstrate the essential files of Langdata folder contains

In Langdata there is also a folder called "BEN". The "BEN" folder contains the main things like our training_text files, config file, font file etc.

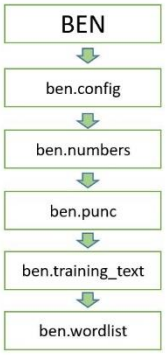

Figure 6.10.3: Demonstrate the essential files of "BEN" folder contains

After creating the corpus successfully. Now we are ready for training. For training purpose, we have to run bunch of commands in Linux. Below we give the commands step by step.

Step 1:

```
sudo src/training/tesstrain.sh --fonts_dir
/usr/share/fonts --lang ben --linedata_only   --
noextract_font_properties --langdata_dir ~/
BengaliTrain /langdata --tessdata_dir
/usr/local/share/tessdata --output_dir ~/
BengaliTrain /bentrain --maxpages 100
```

After executing this command, the following in printed out after a successful run:

```
Created starter traineddata for LSTM training of language 'ben'
Run 'lstmtraining' command to continue LSTM training for language
```

Step2:

```
sudo src/training/tesstrain.sh --fonts_dir /usr/share/fonts --lang ben --linedata_only  --noextract_font_properties --langdata_dir ~/BengaliTrain /langdata --tessdata_dir /usr/local/share/tessdata --maxpages 100  --fontlist "FreeSans" "FreeSerif" "FreeSerif Italic" "Siyam Rupali Regular" "Kalpurush Regular" "Lohit Assamese" "Lohit Bengali" --output_dir ~/BengaliTrain /beneval
```

Here we include all the fonts that we installed for training purpose. We total use seven fonts for training purpose, and they are:

- Free Sans.
- Free Serif.
- Free Serif Italic.
- Siyam Rupali Regular.
- Kalpurush Regular.
- Lohit Assamese.
- Lohit Bengali.

Before running step 2 commands, first we have to add font properties information in our corpus folder font properties file like:

```
Kalpurush_Regular 0 0 0 0 0
Siyam_Rupali_Regular 0 0 0 0 0
```

And also add font information in language-specific.sh file like:

```
'Kalpurush Regular'
'Siyam Rupali Regular'
```

Step 3:

Here we create a folder named benoutput, basically this folder contains our all-base train log information like:

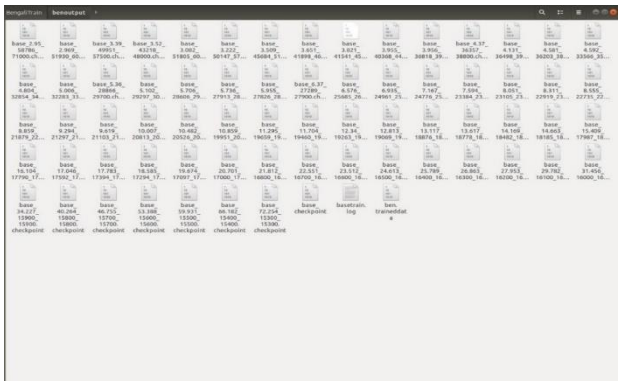

Figure 6.10.4: Benoutput Folder Sample

To create benoutput folder run this command:

```
sudo mkdir –p~/BengaliTrain /benoutput
```

Step 4:

```
sudo lstmtraining --debug_interval -1 --traineddata ~/BengaliTrain /bentrain/ben/ben.traineddata --net_spec '[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c111]' --model_output ~/ BengaliTrain /benoutput/base --learning_rate 20e-4 --train_listfile ~/BengaliTrain /bentrain/ben.training_files.txt --eval_listfile ~/BengaliTrain /beneval/ben.training_files.txt --max_iterations 5000 &>~/ BengaliTrain /benoutput/basetrain.log
```

If you have a GPU in your system, then use –debug_interval 1 or any positive number. If you don't have any GPU, then use negative number. In our case we don't have any GPU that's why we use –debug_interval -1. Also, you have to run this command over and over if you want to reduce the error rate in your trained-data. Cause after executing the command you will see that you got 99.999 error rate.

So you have to run this command over and over. To do this just increase the number of –max_iterations. We found 100000 iterations is the decent level where your trained data perform well. In our case we trained over week and every day we increase 4000 iterations. Cause electricity problem and also our system can't handle too much iterations. As we mention to training tesseract you need a powerful system.

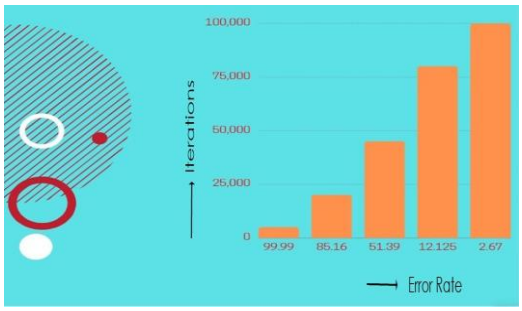

Figure 6.10.5: Iteration and Accuracy Ratio

After executing the command, we can see how our training goes. We observe that at 15000 iterations the trained-data starting recognizing. Here we give a sample of our basetrain.log file.

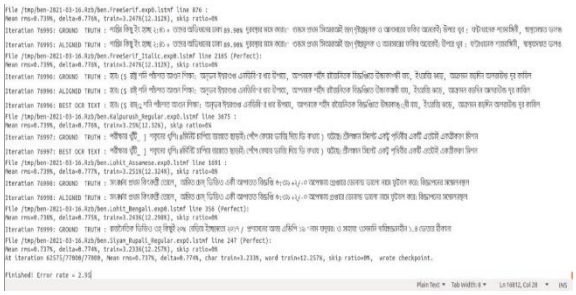

Figure 6.10.6: Demonstrate our Base Log File

Here as we can see we are at now 77000 iterations and got 2.95% error rate. We have to reduce the error rate to get a decent trained-data.

Step 5:
When we achieve our goal, and see that our trained-data perform well, and also, we get less error rate we stop the training. And create the trained-data. To do this we have to run this command.

```
sudo lstmtraining --stop_training --continue_from ~/BengaliTrain /benoutput/base_checkpoint --traineddata ~/BengaliTrain /bentrain/ben/ben.traineddata --model_output ~/ BengaliTrain /benoutput/ben.traineddata
```

6.11 Accuracy Rate

After all the work done when we want to check that how our trained-data perform. We found it perform very well. Which is very good sign for us? Because if you give any image (as input) and want to extract the image as text or want to convert PDF to DOC. Our trained-data will give you the output with 98% accuracy rate. And that's how a good OCR should perform. Below we provide some sample.

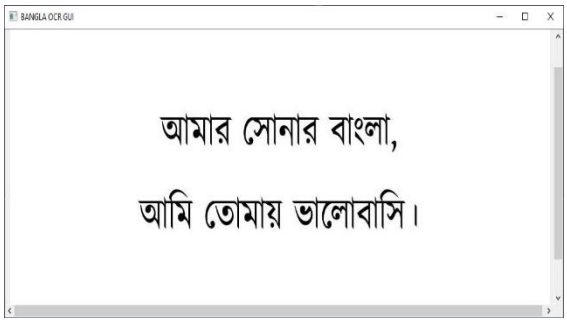

Figure 6.11.1: Random with Bengali Text Written on it

As we can see we give input a random image in our app to extract text form it. And after extracting we get text as an output in Figure 6.11.2. If we compare the input image and output text, it's completely matching with each other. But if we extract large amount of data, we get error, as we mention earlier our trained-data gives 2% error rate.

Figure 6.11.2: After extracting the image.

It detects the space and new line perfectly. Letters correction is good. It can detect the modifier as well as the punctuation. As we said earlier with our LSTM traineddata we achieve 98% accuracy rate. After doing all these training procedures we ensure that LSTM training is far more suitable for Indic language compare to Legacy training.

7.1 New Font Installation

Tesseract by default give some font to use to create traineddata, but the font is too old to use. So, in modern time we need new font. In our training process we use different kinds of modern fonts like Kalpurush Regular, SiyamRupali Regular etc. To use these fonts, we have to do some steps first steps will be download these fonts from internet. There are many websites that provide different types of modern fonts. After downloading these fonts, we unzip them and move the font to usr/share/fonts these folders. To do this we use this command.

```
Sudo KalpurushRegular ~usr/share/fonts
```

After moving the files into the fonts folder then we open the KalpurushRegular folder and install it. After installing we have to add those fonts name in languagespecefic.sh file. The languagespecific.sh file is in Tesseract/src/training folder.

```
BENGALI_FONTS=( \
    "FreeSans" \
    "FreeSerif" \
    "FreeSerif Italic" \
    "Siyam Rupali Regular" \
    "Kalpurush Regular" \
    "Lohit Assamese" \
    "Lohit Bengali" \
)
```

Figure 7.1: Example of Language Specific Shell File

And then the last step will be that, we have to add font properties information in our corpus folder like –

```
Kalpurush_Regular 0 0 0 0 0
Siyam_Rupali_Regular 0 0 0 0 0
```

7.2 Problem Encounter and Solve

When we run our first command to create LSTMF file from our langdata folder. We encounter a unicharset problem and the problem looks like:

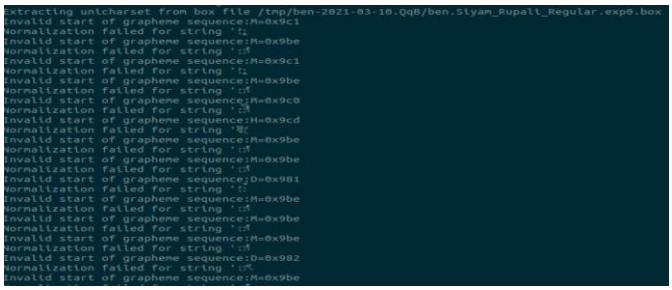

Figure 7.2: Unicharset Problem

After days of researching, we resolve the problem. In this topic we also talked with Tesseract Forum and they also help us. To resolve this problem all we have to do this. First go to this folder src/training/tesstrain_utils.sh. Then Change,

```
--norm_mode “${NORM_MODE}”${box_files}
To
--norm_mode “${NORM_MODE}”${TRAINING_TEXT }
```

8.1 Comparison between Legacy and LSTM Training

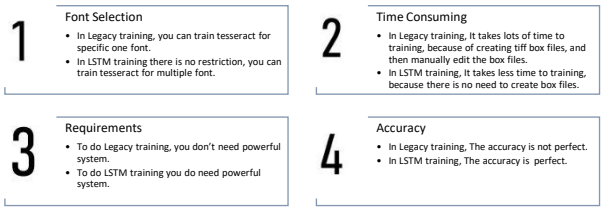

Figure 8.1: Comparison between LEGACY and LSTM

8.2 Accuracy Ratio between Legacy and LSTM Training

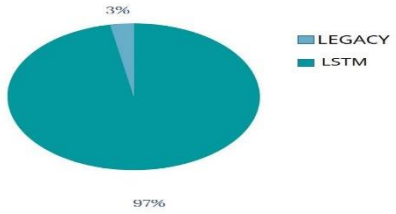

Figure 8.2: Accuracy Ratio between Legacy and LSTM

9. SYSTEM IMPLEMENTATION

Our first priority is creating a GUI app so a non tech user can also use our app. putting multiple features in one platform like - Image to Text convert, PDF to Word convert. Also make sure the accuracy rate is up to mark. And last but not least make sure that our app stays open source so that's why many people can use our app in daily basis. We implement our app using python.

9.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- Tkinter.
- WxPython.
- JPython.

For our OCR development we choose Tkinter GUI.

9.2 Tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Example:

```
#!/usr/bin/python
import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
```

9.3 Qt Designer

Qt Designer is the Qt tool for designing and building graphical user interfaces. It allows you to design widgets, dialogs or complete main windows using on-screen forms and a simple drag-and-drop interface. It has the ability to preview your designs to ensure they work as you intended, and to allow you to prototype them with your users, before you have to write any code. Qt Designer uses XML .ui files to store designs and does not generate any code itself. Qt includes the uic utility that generates the C++ code that creates the user interface.

Qt also includes the QUiLoader class that allows an application to load a .ui file and to create the corresponding user interface dynamically.

PyQt does not wrap the QUiLoader class but instead includes the uic Python module. Like QUiLoader this module can load .ui files to create a user interface dynamically. Like the uic utility it can also generate the Python code that will create the user interface. PyQt's pyuic4 utility is a command line interface to the uic module.

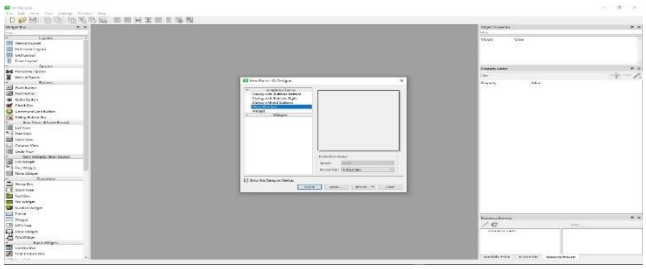

Figure 9.3: Interface of QT Designer Application

9.4 Interface of our software

Figure 9.4: Interface of Our Software

When anyone run our program, first this interface window will pop up in computer screen. As we all can see in this interface there are two button one is IMAGE to TEXT converter and another is PDF to DOC converter.

When anyone click the first button it will pop up another GUI screen where we can convert any Bengali image to text. Whenever anyone press the second button it will pop up another GUI screen where you choose a Bengali pdf and our converter will convert the pdf into doc file.

9.5 Image to Text Converter UI

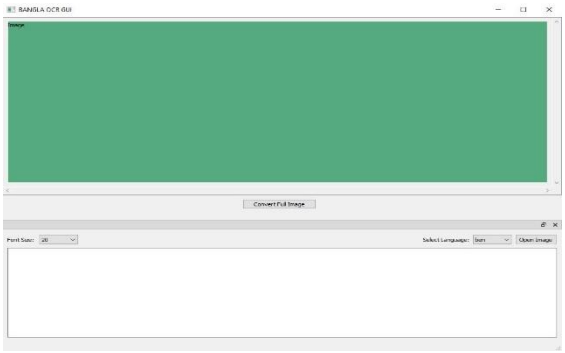

Figure 9.5: Image to Text converter

As we mention in the earlier about Qt designer, we design our whole "Image to Text" converter GUI using Qt designer. Where we have to do just drag and drop for the interface, and the designer will generate the code for us. But we have to write code for the logical execution.

As we can see in right side, there is a button called "Open Image". When someone click that button. It will prompt a window where we can choose any image, as our OCR supports only Bengali Language so we have to choose an image where Bengali words are written. When we choose the image, we can see the preview of that image in the green area. In the middle section, we can see another button called Convert full image. When we click that button it will convert the whole image and give the output in our text area. In our text area there is an option, where you can control the font size. In our "image to text" app we add a special feature. Suppose someone choose an image, then he or she want to convert a small portion of that image, so they can select which portion he or she want to convert using mouse then our converter will convert that small portion and gives the output in the text area. Below we provide our "Image to Text" converter code.

9.6 PDF to DOC

PDF to Doc is the second option available in our converter, which we shown earlier in our interface Fig [9.4]. So the basic operation is whenever someone click the "PDF to DOC" button it will open a prompt window where we can choose any Bengali pdf we want, there is no limitation that you have to choose a pdf that contains only 10 pages, there is no page limitation.

After choosing the PDF our converter will convert the pdf pages into images and stored the images into a local storage. And then our converter converts the images into text, and merged all the text start to bottom into a doc file. And give the output as a doc file in UTF-8 encoding.

To make it simple we don't create any interfaces for "PDF to DOC" converter. You just have to choose the PDF and our converter will do the rest of work. That's why it's more user friendly. Below we provide our "PDF to DOC" converter code.

CONCLUSION

The OCR is implemented on research purpose. It has got its limits but it is an example of training an engine with expertise. We went through a lot of trial and error processes to develop the better Bengali OCR system. As a result, we came up with a better solution, the LSTM training. We assume that we are the first one who trained tesseract for Bengali languages along with multiple fonts using LSTM training. And also our trained data achieve more than 97% error correction rate. We used total six fonts to develop the character set for the OCR. So, the OCR would work fine on inputs that contain text in that font. However, if there are texts of other fonts available in the input, the OCR may mismatch to some extent. Though it is not quite possible to incorporate all fonts of Bengali language to the OCR due to System limitation. Also we make an app using python providing various feature at free cost, In future this project will be open for general purpose use for common users after adding further improvements.

10.1 Limitation

- It cannot convert those image or pdf. Which has graphical image in it. So to use our OCR app we need fresh background image. Unless it cannot convert it.
- If you give lots of data to convert. It makes the process slower.
- Although we got 97% error correction rate. Although when our OCR app cannot convert any word or sentence it gives garbage value.

10.2 Future Works

- In future we want to work with more fonts.
- Also, we want to continue the training to until we gained error correction rate 0.1% or 0.2% something.
- In future this project will be open for general purpose use for common users after adding further improvements.

References

[1] S. Manzur, S. Islam, A. Foysal and A. Chowdhury, "Bangla character recognition for Android devices", *Hdl.handle.net*, 2015. [Online].

[2] M. Chowdhury, M. Islam, B. Bipul and M. Rhaman, "Implementation of an Optical Character Reader (OCR) for Bengali language", *Hdl.handle.net*, 2016. [Online].

[3] R. Smith, "An Overview of the Tesseract OCR Engine", *Ieeexplore.ieee.org*, 2021. [Online]. [Accessed: 11- May- 2021].

[4] S. Bhowmick, *Optical Character Recognition for Bengali Language using Tesseract*, 1st ed. Stamford University Bangladesh, 2018.

[5] M. Hasnat, M. Chowdhury and M. Khan, "An Open Source Tesseract Based Optical Character Recognizer for Bangla Script", *Ieeexplore.ieee.org*, 2021. [Online].

[6] S. Arif, "Bengali Character Recognition using Feature Extraction", *Hdl.handle.net*, 2007. [Online].

[7] R. Smith, "A Simple and Efficient Skew Detection Algorithm via Text Row Algorithm", *Computer.org*, 2002. [Online].

[8] A. Abduallah and M. Khan, "A survey on script segmentation for Bangla OCR", *Hdl.handle.net*, 2007. [Online].[Accessed: 16- Feb- 2021].

[9] B. Chaudhuri and U. Pal, "An OCR system to read two Indian language scripts: Bangla and Devnagari (Hindi)", *Ieeexplore.ieee.org*, 2002. [Online]. [Accessed: 20- Apr- 2021].

[10] Tessdoc,"*Tesseract Documentation"*. [Online]. [Accessed: 11- Apr- 2021].

[11] Rakshit, S., Ghosal, D., Das, T., Dutta, S., "Development of a Multi-User Recognition Engine for Handwritten Bangla Basic Characters and Digits",arXiv.org, 2009. [Online]. [Accessed: 16- Feb- 2021].

[12] S.V. Rice, F.R. Jenkins, T.A. Nartker, The Fourth Annual Test ofOCR Accuracy, Technical Report 95-03, Information Science Research Institute, University ofNevada, Las Vegas, July 1995. [Accessed: 22- Feb- 2021].

[13] R.W. Smith, The Extraction and Recognition of Text from Multimedia Document Images, PhD Thesis, University of Bristol, November 1987. [Accessed: 16- Feb- 2021].

[14] G. Nagy, "At the frontiers of OCR", Proc. IEEE 80(7), IEEE, USA, Jul 1992, pp 1093-1100.

[15] G. Nagy, Y. Xu, "Automatic Prototype Extraction for Adaptive OCR", Proc. of the 4th Int. Conf. on Document Analysis and Recognition, IEEE, Aug 1997, pp 278-282.