Bangla Optical Character Recognition from Printed Text using Tesseract Engine

by

MD Yamin Faruque
16201059
MD Zahin Adeeb
19241013
Muhammad Maswood Kamal
16201038
Redwan Ahmed
16241005

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
BRAC University
January 2021

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

[illegible]	[illegible]
MD Yamin Faruque 16201059	MD Zahin Adeeb 19241013
Maswood Kamal	[illegible]
Muhammad Maswood Kamal 16201038	Redwan Ahmed 16241005

Approval

The thesis/project titled “Bangla Optical Character Recognition from Printed Text using Tesseract Engine” submitted by

1. MD Yamin Faruque (16201059)
2. MD Zahin Adeeb (19241013)
3. Muhammad Maswood Kamal (16201038)
4. Redwan Ahmed (16241005)

Of Fall, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on December 26, 2020.

Examining Committee:

Supervisor:
(Member)

Hossain Arif

Hossain Arif
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Dr. Mahbub Alam Majumdar
Professor and Chairperson
Department of Computer Science and Engineering
BRAC University

Ethics Statement

We, hereby declare that this thesis is based on the results we obtained from our work. Due acknowledgement has been made in the text to all other material used. This thesis, neither in whole nor in part, has been previously submitted by anyone to any other university or institute for the award of any degree.

Abstract

Optical Character Recognition or OCR is a technology that enables us to detect and extract text from images. In our project, we are designing our OCR system around the Bangla language. This is primarily because, there are many models of text recognition of the English language in the market but there are very few on Bangla. Our proposed system comprises of acquiring the input image, pre-processing it, passing it into the Tesseract OCR engine (the backbone of our system) and finally getting digital output of the text. We have used the latest version of Tesseract, that is, version 5 and even though this is in its alpha stage, it is still stable for end-users. Next, to improve accuracy, we have focused on pre-processing the image as thoroughly as possible and laid out our chosen algorithm in each step. For example, for binarization, we have used Otsu's Thresholding algorithm as this gave us the best results. For segmentation, we have used the Fully Automatic Page Segmentation from Tesseracts own repertoire of segmentation modes. Then we have done our training through Tesseract's new LSTM engine and improved upon their existing trained file with our fonts. We have selected these fonts based on their popularity of use. We calculated our accuracy at the word level and our model gave us an average accuracy of 95.9% on multiple fonts and on multiple real life scenarios. At best case scenario we have even managed to secure 100% accuracy. Finally, we have discussed future improvements like the addition of a custom dictionary in our model and how it would increase the overall accuracy in all cases.

Keywords: Optical Character Recognition, Bangla Language, Bangla OCR, Tesseract, RNN, LSTM, Open CV, Otsu's Thresholding Algorithm, Python, jTessEditorFX, Image Processing, Custom Dictionary.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.

Secondly, we thank our supervisor Hossain Arif for giving us this opportunity to research on this topic and assisting us through the process. We would like to express our special gratitude and appreciation to him for giving us such attention and time. We would also like to give our thanks for giving us time to discuss about the topic and referring us to resources and research materials for our Thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

At first, we have described the problem statement to layout an idea about the importance of our project. Following that we have our suggested approach to this issue. Then we have talked about some related works that have previously been done and served as an inspiration to our project. After that, we discussed about the various intricacies that we had to take into consideration regarding the Bangla language. Then, we moved on to talking about all the factors that can affect the output accuracy of our results and laid out available solutions and our selected approach for each issue. After that, we have given an overview about our methodology and the Tesseract engine. Up next, we have talked in details and laid out a step by step guide to our approach and finally concluded with results and analysis of our project along with suggestions for future improvements.

1.1 Problem Statement

Data and our ability to store, maneuver, manage and make use of this data in the most efficient manner will help us achieve all sorts of feats in this digital era. That is why we chose to base our thesis project on OCR technology. Innumerable data are being generated manually by hand everyday for office work or other purposes. These then may need to be inputted into the system for various purposes, which can consume a lot of time if the data is very heavy. The use of computer and its fast processing capability is the most effective tool we have in hand to make use of data in accordance to our needs. So, one needs an effective and simple mechanism that would allow for us to transfer data that is in non-digital format, to digital format. A well trained Optical Character Recognition system is the answer in these cases. Now, there are many OCR systems available for English language in the market. Hence,the digitalization of English based text falls in the norms. But for Bangla, it's not as readily available.

1.2 Solution

Bangla is a language that is used by more than 400 million people, approximately. And our goal is to create a model that would help reach these people the benefits that come with digitalization of Bangla texts. The benefits include, reading aid for the blind, automatic text entry into the computer for desktop publication, library cataloging, ledgering, automatic reading for sorting of postal mail, bank cheques and other documents, document data compression, language processing such as indexing, spell checking, grammar checking and the list of possibilities go on. There are no models for Bangla that are capitalizing on so many practical applications and our vision is to venture towards a model that can be a starting point of Bangla text recognition being readily available for use for the mass. So as a starting point, we want to design a model that can recognize printed text and output an editable and digitally storable version of that text for the Bangla language. Not only that, we also want our system to be accurate in multiple trending Bangla fonts and show near perfect accuracy for the most popular font based on our research.

Chapter 2

Related Work

The importance and use of OCR are increasing day by day due to technological advancement[3]. Some practical use cases could include reading aids for the blind, automated data entry, library cataloging, ledgering, compression of data, indexing, spell checking, grammatical checking, etc. Due to the rise of such applications, research focusing on Bangla OCR was done recently to shed more light on this topic as few have done so.

SK Alamgir Hossain and T. Tabassum in their paper "Neural net based complete character recognition scheme for Bangla printed text books" discussed how their system was specialized for detecting printed text more efficiently[14]. The rotation invariant capability of their OCR system made it special and stand out when compared to other OCRs. The system consisted of mainly 4 stages. They were pre-processing, segmentation, training/recognition, and lastly post-processing. The process begins by pre-processing the input image through binarization, rotation, scaling, and noise elimination. This is because the input image may have noise, improper scale, or rotation, therefore these issues must be eliminated before proceeding with the segmentation stage. This input image is obtained using a flatbed scanner as a grayscale image. During binarization, the grayscale image is converted to a black and white image where each pixel either has a value of 1 or 0, so no gray value or in between is allowed. Then this image is rotated and aligned properly. This is because it's very common for the input image to be slightly rotated which usually causes OCRs to have trouble recognizing characters. One of the difficult tasks for them was to find the exact degree at which the image was rotated, which they solved by creating an algorithm specialized for it. It was then later scaled properly to a ratio of 1 to 0.8, which is the ratio of standard letter-size pages. It was then segmented and the individual characters are extracted out from the input image. In this stage, the texts are first segmented into lines which were then transformed into words, which were then later transformed into characters. These were forwarded to the system so that they can be trained and for the characters to be recognized. The normalized vector of each character, which is 256-bit long, obtained from the training dataset was used to train this neural network. This network primarily consisted of four neural network layers along with two hidden layers to enhance the classification ability. The network is then trained with random data so that the weights are updated. During the recognition phase, the confidence value is obtained from just one single iteration of each class of the character set. With the help of the 256-bit vector we got previously, if the output of the system matches with one of the characters with a certain amount threshold, then that specific character is recognized. However, if it is outside that specific threshold, then the system is unable to detect the character and hence returns 0, representing its inability to recognize the character properly. After training and recognition, the next step for the system was to reconstruct the original texts with the help of Bangla word formation rules[2]. After rigorous testing and experimentation, it was found out that the character segmentation success rate was 81 percent, word segmentation success rate was around 96 percent, and lastly, the line segmentation rate was determined to be 98.8 percent. For testing, the authors used 32 pages from the novel "Pather Panchali"[1]. "AdorshoLipi" was used for the font type. Fonts like "Sulekha", "Rupali", and "Siyam" showed very good recognition success rates when compared to other fonts upon further analysis. The main goal of the writers was to build an OCR system for printed textbooks that would be rotation invariant while also providing decent or satisfactory results. Based on the results they got from the test cases, they were able to confirm that the system had the potential to digitize text for Bangla books.

In the paper "An open source Tesseract based Optical Character Recognizer for Bangla script" the authors Md. Abul Hasnat, Muttakinur Rahman Chowhudry, and Mumit Khan discussed how they built a Bangla OCR from scratch by using both Tesseract and BanglaOCR engine[8] [5]. It mainly utilizes the best features of both Tesseract and BanglaOCR, which is the incredible recognition ability and script processing power respectively. The authors primarily focused on preparing the training data, integrating the Tesseract engine, and the post-processing procedures for the OCR in this paper. BanglaOCR is a full-fledged OCR framework with a recognition rate of up to 98 percent. However, it is a challenge to achieve this rate as it is heavily dependent on the document type, typeface, font size, and a large training dataset. On the other hand, "Tesseract based BanglaOCR" is an OCR software that is combined with BanglaOCR which takes advantage of Tesseract's incredible character recognition engine. It consists of a GUI that enables the user to input an image, detect the characters in the image, and then provide suggestions to the user to improve the accuracy of the results. The complete methodology of this system consists of 5 primary steps, which are preparing the training data, pre-processing the aforementioned training data, assembling those data for it to be supported by Tesseract, identifying the characters with the help of Tesseract engine, and lastly outputting the final generated text after post-processing. The authors initially followed the guidelines described in[9] to prepare the training data for Bangla script. At first, 340 characters were listed which were considered to be the primary training units. The authors had to do a lot of testing and experimentation to figure out the right combination of training data that would yield them the highest accuracy. In the end, they had to prepare 14 datasets of training data with 5 parameters: the type of the document image, type and size of the fonts used, DPI information of the image, segmentation, and lastly degradation. The reason why such large data sets were created was because of Tesseract's limitation of only supporting 32 configuration files for every unit[10]. The values that these parameters would hold were all very crucial since all of the training images were auto-generated in the data preparation phase. The font type "SutonnyMJ" was used for these combinations with a size of 24 and a DPI of 300. The primary objective of the pre-processing step was to get information about the characters after segmentation was applied. The ability to read any format of input image was a vital part of the system, therefore, acquiring images and the process of extracting data from those images were all very important. In order to finish the remaining tasks of the pre-processing stage the authors used the techniques at[9]. All the character information were later stored in an array at the end of this stage. Since Tesseract is only capable of reading uncompressed 1bit/8bit tiff encoded images, the generated images also had to be encoded in that format for them to be supported and readable. These images were generated, using segmentation data from previous steps, and were then temporarily saved. The temporarily saved images were then used and processed in the recognition phase of the Tesseract engine to obtain the output texts. The final post-processing stage was composed of two levels, the first level used the raw text output from the previous stage and the second level used the output from the first stage of the post-processing. The first level was divided into further two separate tasks, where the first task was to reorder the Unicode text from the previous step, and the second task was to correct any known recognition errors by following a set of rules. The second level then tries to spell check the output generated by the first level by using a dictionary that consists of 180k words. The authors followed the post-processor mentioned here[6] to perform the steps of this level. The authors concluded by mentioning the accuracy of the system which was 93 percent for high-quality input images and 83 percent for images that are of poor quality or have small font size.

Sanjit Kumar Saha, M. Shamsuzzaman, and M. A. Bhuiyan in their paper “On Bangla Character Recognition”, talked about how they implemented a hybrid neural network that merged both artificial neural networks and local image sampling for their Bangla OCR[11]. It depended on multi-layer perception and dimensional reduction, therefore BAM was used as a base for it. A backpropagation algorithm was used to train such a network, however, it was found out that their system required a huge amount of iterations to train it to a satisfactory level. The process began by first acquiring Bangla texts using a scanner which was then geometrically normalized and binarized. Afterward, the “matra’ lines were detected and separated. The baseline detection was attained by making use of a special type of histogram construction. This was done by counting the number of black pixels followed by white pixels of a single line. After the words are segmented, a word was taken out from the “matra” to baseline. A horizontal histogram was created, and with the help of this, the number of white pixels that separate each character was found, which also later helped to distinguish between the characters. The hybrid neural network that they used is said to be made of 2 neural networks. They are BAM (Bidirectional Associative Memory) NN and Bidirectional NN. BAM’s main purpose was to speed up recognition and make it more efficient. Consequently, the neural network was then trained to obtain feature specific data for each character. Next, the system was tested with different test patterns to get a performance report of the network, which was later found out to have 98 percent accuracy.

In their paper “Text Pre-processing and Text Segmentation”, Archana A. Shinde and D.G. Chougule discussed the application areas in the implementation of an OCR software[13] and focuses on 2 major functions: an algorithm to correct the skew angle of an input image which may come from scanning, and a segmentation procedure that uses a novel profile-based method which aids in differentiating between lines, characters, and words. Skews are a distortion in the scanned input image which is very common to find in scanned documents. Skew detection and correction algorithms have a significant effect on the character recognition rate, which is why it's essential to correct any skews related errors before proceeding to further stages. This is also one of the main reasons why most document analysis programs today have such algorithms to tackle these issues. In conjunction with their paper, a standard OCR system included collecting images, pre-processing, segmentation, feature retrieval, and lastly recognition. Segmentation is an important aspect of an OCR system since wrongly segmented characters are not adequately recognized. This would cause the recognition rate of the OCR system to fall. Several approaches are available in the literature for image de-skewing. The system mentioned is based on the Fourier transform[3][4]. The input image needed to be first transformed from the spatial domain to the frequency domain. After that, the "frequency domain" was needed to be observed. Projection profiles were used as a base for the segmentation of the proposed algorithm. The binary image was segmented at various levels which included line segmentation, word segmentation, and character segmentation. The approach suggested began by segmenting the lines and words from binarized de-skewed images by making use of a method that involved horizontal and vertical projection respectively. The horizontal and vertical profiles were then determined in the projection profile methods. The vertical projection profile could then be used to segment each word into individual characters. In the pre-processing stage, since Bangla texts are usually a series of letters arranged horizontally and the lines are piled over one another, the authors made use of this assumption and were able to predict and confirm that these lines would be the area where the frequency domain would be the highest. The image would then be captured from a random angle, and this angle was later saved for upcoming testing stages. To minimize the possibility of errors and to get better de-skewing results, the spectrum of this image was segmented into four quadrants, like the four quadrants which represent the Cartesian coordinate system. Around 15 to 20 bright points are found in this spectrum, which the authors used to draw a straight line that would fit and go along those points. This helped them to determine the angle between the x-axis and the line. This same process is applied for the other three quadrants and the angle between the line and the x-axis is determined for each of those quadrants. Next, these angles would then be averaged to evaluate the actual skew angle which would in turn help to obtain the de-skewed image by rotating the binarized image at the averaged angle found from the quadrants. The author then proceeds to the segmentation phase where they implemented their own custom algorithm to make the process of separating words and lines from texts much faster and efficient. This method involved using horizontal projection profiles to separate the words from texts, whereas vertical projection profiles would be used to separate the lines from texts. In this algorithm, segmentation of the lines was done first, and then the system proceeded with the word segmentation. The horizontal projection profile (also known as HPP), which the line segmentation depends on, consists of a histogram that has a series of ON pixels on every row of the image. The word segmentation on the other hand relies on the spaces between the words in the text which is found by utilizing the vertical projection profile (also known as VPP). Similar to the horizontal projection profile, the vertical projection profile is a histogram that consists of the sum of ON pixels on every column of the image. Lastly, all the algorithms were later set up in Matlab and heavily tested with documents that have clean and good image quality and didn't have any overlapping or broken characters that might interfere with the recognition, though this is an area which the authors would like to touch upon in the future to make their system even better.

In the next paper, "A Complete Workflow for Development of Bangla OCR" by Farjana Yeasmin Omee, Shiam Shabbir Himel and MD Abu Naser Bikas, they talk about a more traditional approach towards implementing Bangla OCR[12]. They have laid out the primary steps, that is scanning, pre-processing, feature extraction or pattern recognition, recognition using one or more classifiers and lastly, contextual verification or post-processing. For scanning, a simple hand-held scanner or flat-bed scanner was suggested. Basically, higher the resolution, the better the output will come in the end. Then they talk about pre-processing which comprises the following steps: 1) Binarization 2) Noise Detection and Reduction 3) Skew detection and correction 4) Page layout analysis and 5) Segmentation. They then further talk about various algorithms and techniques for each of the steps. Then they talk about the importance of segmentation, especially in the case of the Bangla language. As we know, Bangla has the matra line which needs to be recognized. Also, we have compound characters, which are a combination of 1,2 or 3 consonants combined together. So the segmentation has to recognize and take care of these intricacies. Hence suitable algorithms and approaches were talked about in their paper. Lastly, they talk about the introduction of the Tesseract OCR engine in the realm of OCR technology and how it eliminates a lot of the steps like feature extraction and pre-processing, that are traditionally required in the making of an OCR system.

Nishatul Majid and Elisa H. Barney Smith presents an offline Bangla handwriting dataset and an algorithm for the recognition of isolated Bangla basic characters in their paper "Introducing the Boise State Bangla Handwriting Dataset and an Efficient Offline Recognizer of Isolated Bangla Characters"[17]. The database consists of 2 pages. The first one has a character essay of 104 words/364. The essay uses 49 simple characters, including 11 diacritics of vowels and 32 conjuncts of consonants of high frequency. The second page includes 84 isolated units with all basic characters, numbers, vowel diacritics, and multiple high frequency conjuncts. The initial release is based on 100 individual authors' voluntary contributions. One of the highlights and unique features of this database is that all its contents are marked from various component hierarchies, such as characters, words and lines, with the associated ground truth information. It is expected to be useful for research on offline handwriting recognition in Bangladesh, especially with segmentation based approaches. Furthermore, a basic character recognition method is presented where the features are extracted based on zonal pixel counts, structural strokes and grid points with U-SURF descriptors modeled with bags of features. The highest classification accuracy obtained with an SVM classifier based on a cubic kernel is 96.8 percent using the isolated characters from the Boise State dataset together with 3 other datasets to ensure the versatility and robustness of this process.

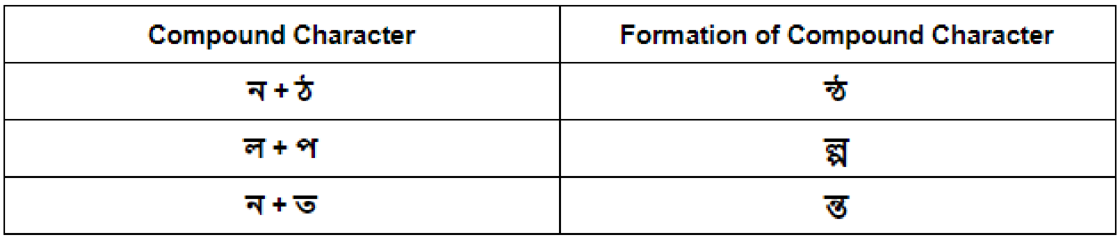
Chapter 3

Special Characteristics of Bangla Text

3.1 Characteristics of Bengali Script

Typical Bangla alphabet has 11 vowels, 39 consonants and 10 basic numerical characters. Bengali font is made up of a complex script pattern as shown in Figure 3.3. There are vowels, consonants and combinations of characters made up of different characters and unclassified characters. In Bangla, each word is connected by a ‘Maatra’. It is a horizontal line at the top that connects all characters but there are some exceptions. Some of the basic bengali script features are mentioned below.

- Bangla is written from left to right
- Bangla does not have lower case or upper case letters.
- Bangla has several short forms of vowels and consonants, such as ‘Kar’, written as ি, া, ু and ‘Fola’ as ক্য, ক্ল
- A compound character consisting of several consonants partly retains the shape of the original characters in a single syllable of a word. (Shown in Figure 3.1 and 3.3)

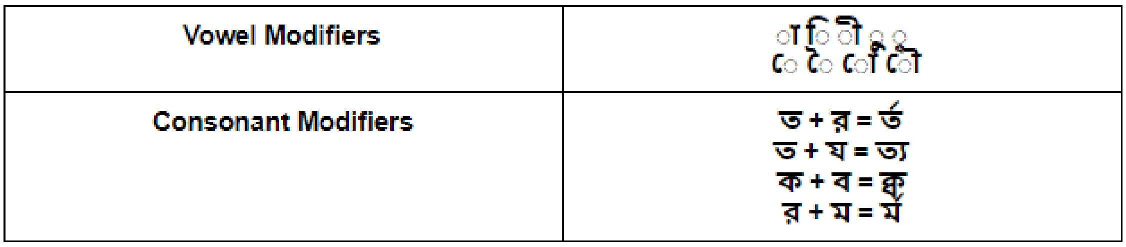


Compound Character	Formation of Compound Character
ন + ঠ	ন্ঠ
ল + প	ল্প
ন + ত	ন্ত

Figure 3.1: Compound Characters

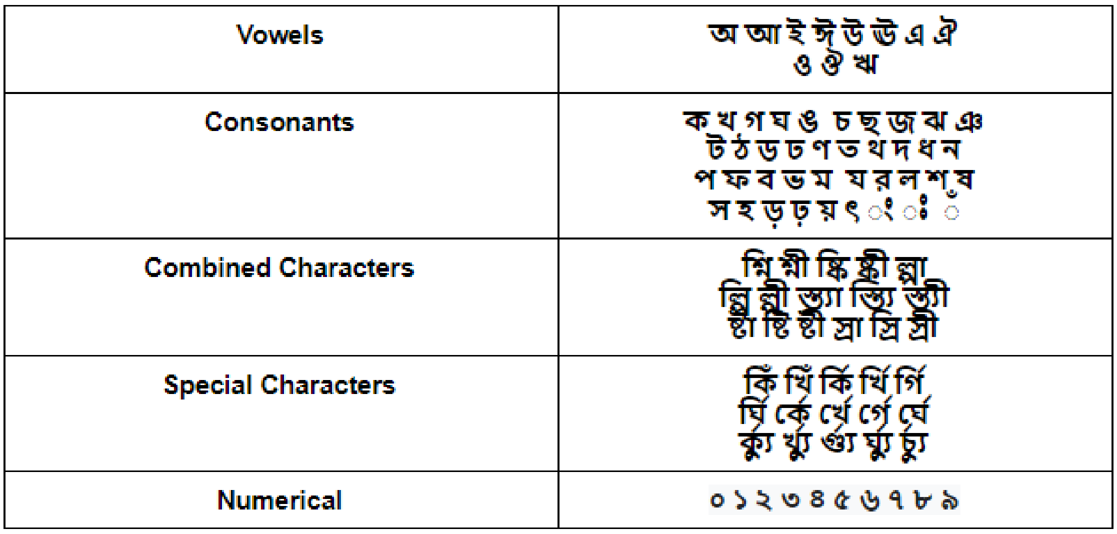
- All characters, except a few (e.g গ,খ,ঙ,ধ) have a horizontal line connecting them called ‘maatra’.
- All characters containing ‘maatra’ stay connected except the one without ‘maatra’ lines.

- Modifiers are vowels which take on a different form in a word. (Shown in Table 3.2)



Vowel Modifiers	া ি ী ু ূ ে ৈ ো ৌ
Consonant Modifiers	ত + র = ত্র ত + য = ত্য ক + ব = ক্ব র + ম = র্ম

Figure 3.2: Vowel Modifiers



Vowels	অ আ ই ঈ উ ঊ এ ঐ ও ঔ ঋ
Consonants	ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড ঢ ণ ত থ দ ধ ন প ফ ব ভ ম য র ল শ ষ স হ ড় ঢ় য় ৎ ং ঃ ঁ
Combined Characters	শ্নি শ্নী ঙ্কি ঙ্কী ম্লা ম্লি ম্লী জ্জ্যা জ্জ্যি জ্জ্যী ষ্টা ষ্টি ষ্টী স্রা স্রি স্রী
Special Characters	র্কিঁ র্খিঁ র্কি র্খি র্গি র্ঘি র্কে র্খে র্গে র্ঘে ক্যু খ্যু গ্যু ঘ্যু চ্যু
Numerical	০ ১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯

Figure 3.3: Sample of Characters

3.2 Properties of Digital Bengali Font

One very important thing to keep in mind is that Bangla font works differently when put into an OCR. For example, the word খা is the combination of খ and the short form of the vowel আ. Short forms are represented as a dotted circle alongside. For example : া ি ী ু ৃ ে ৈ ো ৌ

The OCR has a hard time recognizing these character sets. For this reason alone, researchers need to include all possible patterns of a character in Bangla language in the data set. For example : the character ক can have form like (কি কী কু কূ কৃ কে কৈ কো কৌ)

These combinations when combined with special characters such as 'ঁ' called 'chandrabindu' which when placed on top of other characters change the pronunciation. For these reasons, the digital Bengali font is very different from the traditional handwritten Bangla font. Also for this reason, we have to come up with different ways to solve the problems that arise with digital Bengali font.

Chapter 4

Output Accuracy Factors

In the past few decades,the technology behind Optical Character Recognition has seen drastic improvements, just in terms of sheer capability alone. Advanced machine learning methodologies, more defined algorithms and the existence of higher than ever computing power have enabled these developments in the OCR realm. But, even with all these tools at hand, reaching accuracy levels of around 99 percent falls under the category of an exception and not at all easy to achieve. But we can get closer and closer to these high numbers and pre-possessing our test images play a big role in that.

4.1 OCR Accuracy

OCR accuracy defines the reliability of the system and can be measured on two levels[15]:

- Accuracy on a character level
- Accuracy on a word level

4.1.1 Character Level Accuracy

The norm on judging the accuracy in OCR technology is upon the character level [15]. It translates to how well an OCR system can recognize a character correctly versus the times the character is recognized incorrectly.The accuracy is assessed by taking the output of an image processed by an OCR engine and comparing it to the original version of the text. For example, a 99 percent accuracy means, only 1 out of every 100 characters is prone to error or "uncertain" and gives us a rough idea of how big of a feat it is to achieve high levels of accuracy. Now, the uncertain character can be correct or incorrect. Basically, what it means is, the OCR system was not able to make a final decision about it, despite all the built-in classifiers and voting algorithms. A way to work around this issue in a real life scenario would require a person to ultimately determine whether the word is wrong or right.[15]

4.1.2 Word Level Accuracy

The recognition quality of an OCR system can also be quantified by measuring the accuracy on a world level, instead of character level [15]. This approach is particularly sought after in scenarios, where proper words need to be found, for example, searching a document or a book.

4.2 Improving OCR Accuracy

If we are talking about making improvements in the accuracy of Optical Character Recognition, then there are two main factors that one would need to consider.

Source Image Quality :

The clarity of the original source image is an important constituent in this matter. That is, if a human can perceive the contents of the image clearly and perfectly, then that raises the possibility of good OCR results. But if it happens to be otherwise and the image is of not decent quality, then that would leave the OCR results prone to errors. In short, a better quality source image translates to the system being able to distinguish characters more easily, hence resulting in higher overall accuracy.

The OCR Engine :

It is the responsibility of the engine to recognize the text from whatever input image is being provided to it. Now, there are various OCR engines available in the market, Google Cloud Vision, Tesseract, ABBY FineReader, OCRopus to name a few and these range from free open source engines to proprietary solutions with a significant price tag.

These engines, as they are trying to achieve the same outcome,at times do share in common the type of algorithms used, but each of them do have their own pros and cons as well that comes with them. Choosing the right type of engine is important and mostly depends on specific use-cases and financial budget.

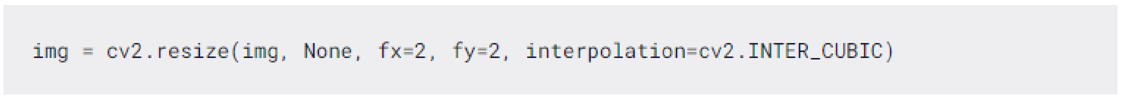
For us, we chose to go with Tesseract as currently it is considered the best open source OCR engine. We will be talking in detail about this engine in another section of this report. For now it is important to note, our reason for choosing this engine was because it is open source, and its developer community is very active in the scene. With proper preprocessing tool-chain, the accuracy of Tesseract can be substantially increased.

4.3 Image Processing

All the input images must be pre-processed before passing it to the Tesseract OCR engine. This stage is mostly about preparing the test images in a format that is suitable for Tesseract OCR to process and is very vital in determining the output accuracy percentage. The output is dependent on the quality of the input images, therefore the resolution of each image should be kept as high as possible. Majority of the top OCR and Automated Forms Processing software companies suggest a minimum resolution of 300 dpi for the most effective extraction of data. In an ideal real life scenario, one would need to take scanned images of the book or samples through a scanner machine. It’s best advised to use a scanner which has an approximate resolution in the range of 300-1000 dots per inch for reliable scan accuracy for extraction. If the images are captured with a mobile phone camera, then the high tier lenses would be able to provide for the best results. At present day, mobile phone cameras have taken generational leaps in terms of quality and hence it is actually a very viable option. Besides the source image being of as high quality as possible, there are some other processing being done through our engine to prepare the best pre-processed input image. Now.we will be talking about some of these pre-processing techniques which were achieved through the help of the Open CV library. [20]

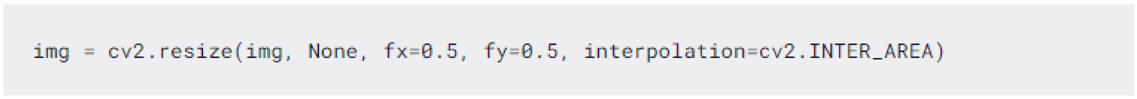
4.3.1 Re-scaling

As we have previously talked about, according to the official Tesseract documentation, their preferred minimum resolution is 300 dpi for the most optimum results. Now, if the input image is lower than the recommended resolution, we can scale it to a larger size. This would enable the engine to recognize small characters, which is very important for the Bangla language. For example, one of the hurdles our machine can face is distinguishing between characters like ব and র [6]. If the image is low resolution, then the dot below র will be considered as a noise and removed in the noise elimination process. To re-scale images, python, our primary choice of language provides us a number of options like INTER_AREA, INTER_CUBIC AND INTER_LINEAR. Upon research and experimentation, we found INTER_CUBIC and INTER_AREA is best suited to upscale an image and INTER_AREA is best when down scaling/shrinking an image.[16]



```
img = cv2.resize(img, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
```

Figure 4.1: INTER_CUBIC command



```
img = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)
```

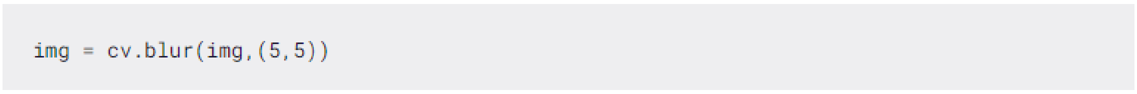
Figure 4.2: INTER_AREA command

The INTER_CUBIC method is relatively slow though, but it gives very accurate results. If speed is an issue, then INTER_LINEAR is a good option, trading off a bit of accuracy for faster performance.

4.3.2 Noise elimination by blurring

When taking pictures with a camera, specially a mobile phone camera, there may be noise added to the image, due to lighting conditions and other factors. Noise may also be produced automatically when scanning documents. The blurring is usually done by convolving the test image with a low-pass filter kernel [16]. Below, we will discuss some methods about blurring to eliminate noise.[16]

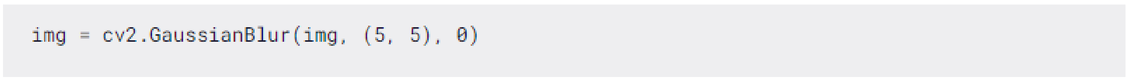
- Averaging : This is a basic blurring technique where the image is convoluted with a normalized box filter by taking a mean of all the pixels under the kernel and replacing the central element.



```
img = cv.blur(img,(5,5))
```

Figure 4.3: Command for Averaging technique

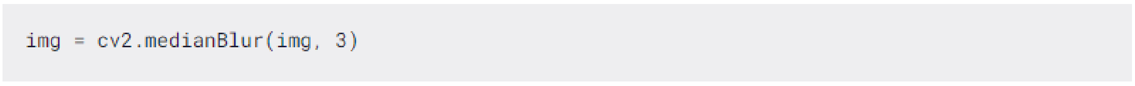
- Gaussian Blurring : This form of blurring functions similarly to the Averaging technique but the difference is, it uses Gaussian kernel instead of normalized box filter for convolution. The kernel dimensions and standard deviations in both directions can be independently calculated. The Gaussian blurring is very effective when it comes to removing Gaussian noise from the image. But on another note, it does not preserve input edges. [16]



```
img = cv2.GaussianBlur(img, (5, 5), 0)
```

Figure 4.4: Command for Gaussian blurring

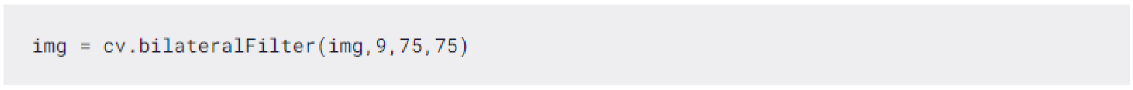
- Median Blurring : One form of noise that may form on the images is known as salt-and-pepper noise. The median blurring technique performs the best when it comes to dealing with this form of noise. It does so, by replacing the central element in the kernel area with a median of all the pixels under the kernel. Moreover, it is noteworthy that this is a non-linear filter. In contrast to linear filters, median blurring substitutes the pixel values with the median value available in the neighborhood values, unlike linear filters. Median blurring thus retains borders, since the value of one of the adjacent pixels must be the median value.[16]



```
img = cv2.medianBlur(img, 3)
```

Figure 4.5: Command for Median blurring

- Bilateral filtering : Finally, we have bilateral filtering. In order to eliminate the noise without smoothing the edges, bilateral filtering gives the best results.This method also uses a gaussian filter to find the gaussian weighted average in the neighborhood. However, while blurring the nearby pixels, it also takes pixel variations into account. This therefore makes sure that only pixels with an intensity close to that of the central pixel are blurred, while pixels with distinct values are not. As a result, the edges that have greater variance in strength, are preserved. [16]



```
img = cv.bilateralFilter(img,9,75,75)
```

Figure 4.6: Command for Bilateral blurring

All in all, go for median blurring or bilateral filtering if you are interested in conserving the edges. Gaussian blurring is, on the other hand, likely to be quicker than median blurring. Bilateral filtering is the slowest of all approaches because of its computational complexity. We can select any of these methods in accordance to our needs.

4.4 Binarization

Next, to help in the recognition process, we need to make it easy for our engine to be able to distinguish between foreground text and background. For that we need to effectively convert our image into a black and white form and this is known as binarization of the image. And the best way to achieve this is through image thresholding. In its simplest form, the basic idea is that we select a threshold value at random and assign it globally for every pixel in the image. If the pixel value is greater than the threshold value, we set the pixel to white. If it is smaller, we set it to black.

Below is a comparison between input and output.

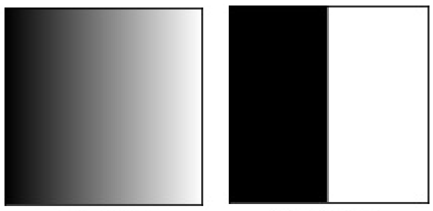


Figure 4.7: Input image on the left and Output image on the right

4.4.1 Adaptive Thresholding

But we also have to keep in mind that, in reality, an image may have different lighting conditions throughout its layout. So, adaptive thresholding would yield better results in such scenarios. The algorithm here calculates the threshold value based on a small region around it. As a result, for different regions of the same image, we get different thresholds and that offer better results for images with inconsistent lighting conditions. This method determines the threshold value in two ways. It can calculate the threshold value as a mean of the neighbourhood area minus a constant, C. The method call would be “cv.ADAPTIVE_THRESH_MEAN_C” for this. Another way would be determining a Gaussian-weighted sum of neighborhood values minus the constant, C. The call for this would be “ cv.ADAPTIVE_THRESH_GAUSSIAN_C”.

The following example has been taken for demonstration purposes from the documentation of Open-CV[20].

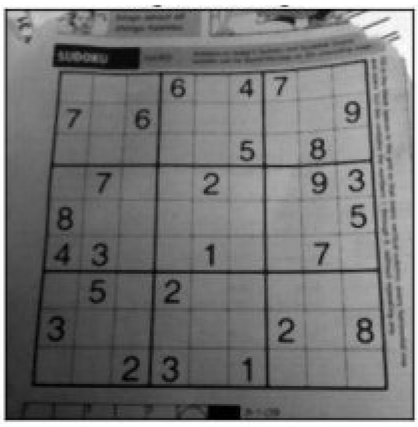


Figure 4.8: Image with varying lighting conditions

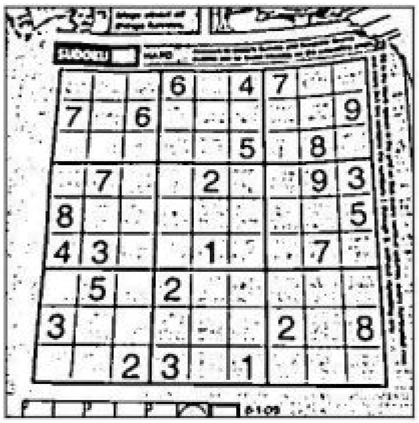


Figure 4.9: After applying Adaptive mean thresholding

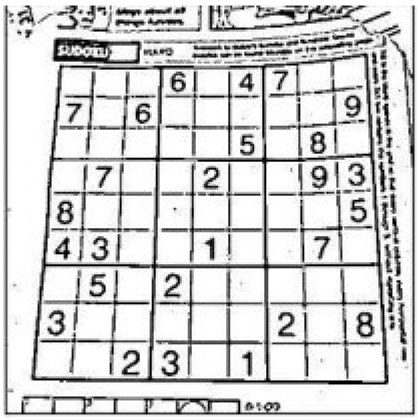


Figure 4.10: After applying Adaptive Gaussian thresholding

4.4.2 Otsu's Binarization

This method was used for our project. What this process does is, it takes an approximate threshold value from the histogram of a bi-modal image. Bi-modal images have histograms with only two peaks, and the threshold value is taken from the middle of those two peaks. This selection of the value is done automatically by the algorithm. Keeping all these in mind, we can tell that if the image is not bi-modal, the Otsu's method will not be accurate. According to the instructions and testing of the Open-CV documentation, they further advised to at first apply a 5x5 Gaussian kernel and then applying Otsu's thresholding. What this does is, it gives a more distinct histogram, hence the algorithm can make a more accurate calculation of the threshold value. The following image has been taken from their documentation. [20]

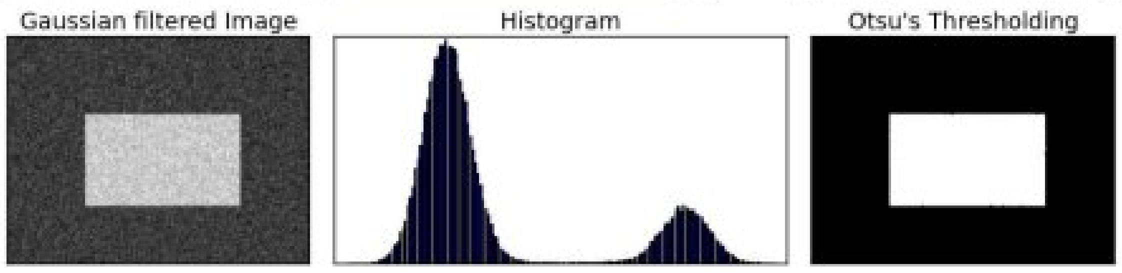


Figure 4.11: Histogram with Gaussian filter

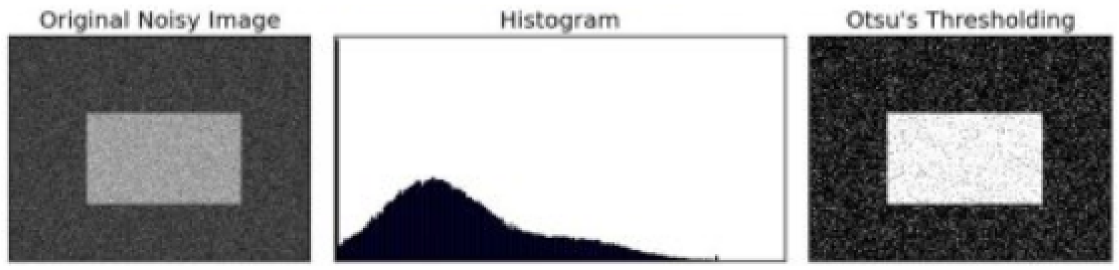


Figure 4.12: Histogram without Gaussian filter

4.5 Further Improvements

Lastly, we will talk about some additional improvements that can be done through pre-processing algorithms, but will not go into algorithmic details as we have manually taken care of these issues rather than through coding. At first, there is the issue of Dilation and Erosion [21]. Various fonts come in various shapes and sizes. Or the document may suffer from heavy ink bleeding resulting in unclear heavy texts. This can prove to be a major issue in recognition especially in the case of the Bangla language, where the sentence structure is already complex. If the font is too heavy, then Erosion techniques can be used to shrink the text and if the text is too thin to be recognised correctly then Dilation techniques can be used to enhance its size [21]. And then another major factor that needs to be handled is the alignment of the text. If the scanned image is not straight this would greatly deteriorate the output accuracy. This needs to be addressed beforehand so that text lines are always horizontal.

Chapter 5

System Overview

5.1 Characteristic Features of the System

This paper offers a variety of features both old and new that helps in training and using a machine learning engine for language and image processing.

- The system has been implemented with a lot of thought about the Bangla language and its quirks.
- It is very easy to train and implement.
- It is very user friendly as it can be implemented and used in the Windows OS.

5.2 Overview of the System Procedures

The system uses a Tesseract OCR Engine with the Command Line Interface. The engine uses a Long Short-Term Memory Recurrent Neural Network, which is significantly better at processing and recognizing images for text extraction than the engines used in the previous versions of Tesseract [7]. The OCR needs only one file with the extension “.traineddata” to be able to recognize and segment the text from an image. This file is the culmination of several different files packed together. After this file has been put in a specific location in the Tesseract directory called “tessdata”, the OCR engine can use this file to uniquely identify each character. If the aim is to get better accuracy in recognition of these characters, then the “.traineddata” needs to be rich and has to have a deep dataset. The Bangla language is a complicated language, with its vowels, consonants and modifiers, and the different combinations and permutations of these base characters [6]. We have trained our engine to recognize the “Siyam Rupali”, "Sagar" and "Akaash" Bangla fonts, for each of the basic characters and its combinations. We have acquired and developed a huge dataset containing all the combinations of each vowel and consonant. It should also be mentioned that the dataset needs to contain every amalgamation of the characters for the OCR to be able to recognize it with a good amount of accuracy and precision [6] .

The system uses a scanned image from a printed book or newspaper to recognize the characters, so “.tif” formatted images were made of the characters used and their combinations, for the font used in the scanned images first. Then we used these images to generate box files for each image. We then generated an ".lstm" file from the existing ".traineddata" file. Then using the ".tif" images and their respective box files, we created a ".lstmf" file for each respective image. We then ran the command for training the OCR engine, which at the end gives us a checkpoint file but this is for the font that we trained the OCR on. We then use the checkpoint file to generate a new ".traineddata" file for our font and combine the old and new ".traineddata" files. The old ".traineddata" file is then replaced with the new and combined ".traineddata" file in the Tesseract directory. All of the training part was done in CLI on Windows Command Prompt. This can be done in both Windows and any Linux distribution but Windows is much more user friendly. We then used VSCode and a python script to run our images for recognition and extraction of text, which gives us an output in the form of an editable text file.

In our paper, we did not train the OCR from scratch but just to add support for a new Bengali font [21]. This training process is called Fine-Tuning in Tesseract, where we train an already trained language with our additional data, which in this case is a font[21]. This process might work with the existing training data file if the font is different from it in a subtle way but would be hard to train for a font in this process, if the font is really unusual. There are other methods of training the engine but they are a daunting task in their own respective ways and they also require the engine to be trained from either scratch or from a more deeper level.

The overall training process is conceptually similar from Tesseract ver. 3.04, but there are some key differences. One of them being, the boxes of the ".box" files now tend to be textline level so that it is far easier to make training data from an image data. Secondly, the ".tr" files have been replaced by ".lstmf" files. Furthermore, different fonts of the same language can now be mixed together instead of them being separate and this is what we should do from on[21]. The different steps required for clustering has now been replaced by a single lstmtraining command line given in the command prompt. Moreover, new performance upgrades and some quality of life upgrades have also been implemented in Tesseract 5 so it now performs much better than Tesseract 4. The engine has also been improved upon since Tesseract 4 as well.[21]

The accuracy of the output, that is to say, how many words or characters the OCR recognizes correctly depends on how well trained the engine is and on the quality of the image used for recognition and training. If the input file is taken from a camera, it is essential to look for any shadows in the image. This is because the OCR engine will have trouble differentiating between the text in the image and the shadows. We get a higher quality image when the input file is scanned from a book or any printed medium. We have to make sure that the resolution of the scanned images is 400 pixels or more for better efficiency.

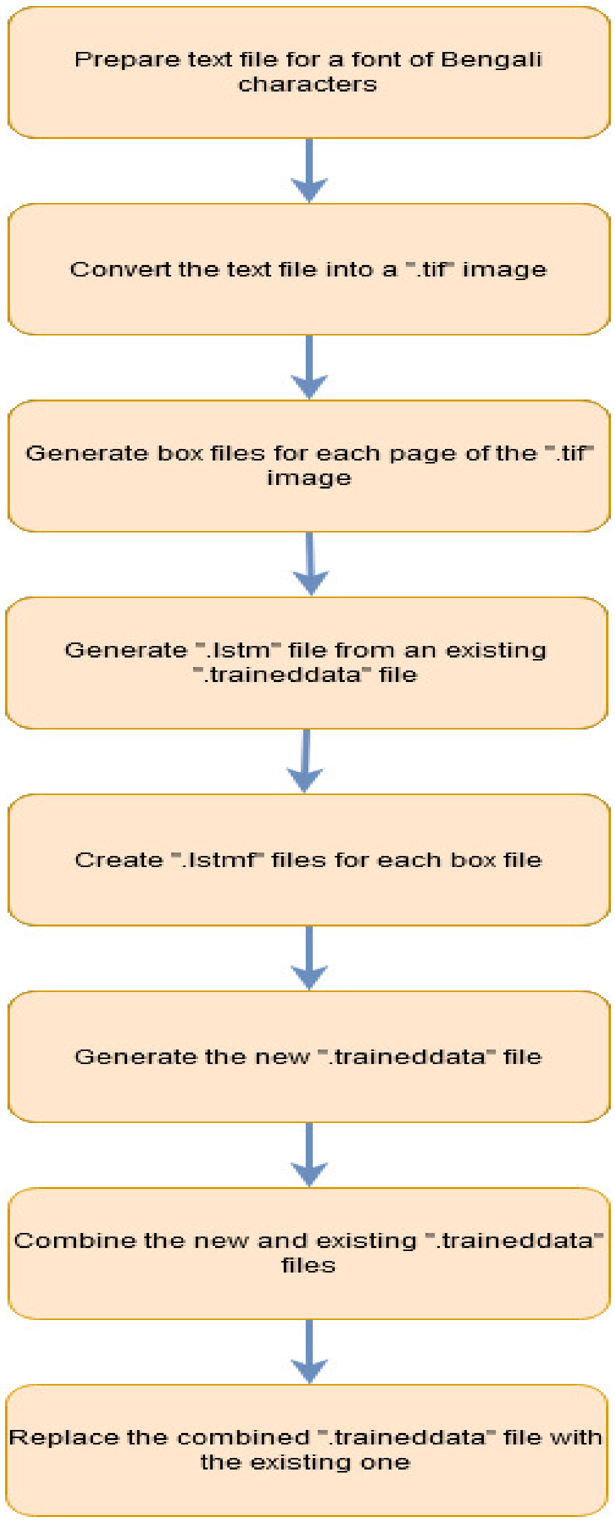

Prepare text file for a font of Bengali characters
Convert the text file into a ".tif" image
Generate box files for each page of the ".tif" image
Generate ".lstm" file from an existing ".traineddata" file
Create ".lstmf" files for each box file
Generate the new ".traineddata" file
Combine the new and existing ".traineddata" files
Replace the combined ".traineddata" file with the existing one

Figure 5.1: Block Diagram of System Procedures

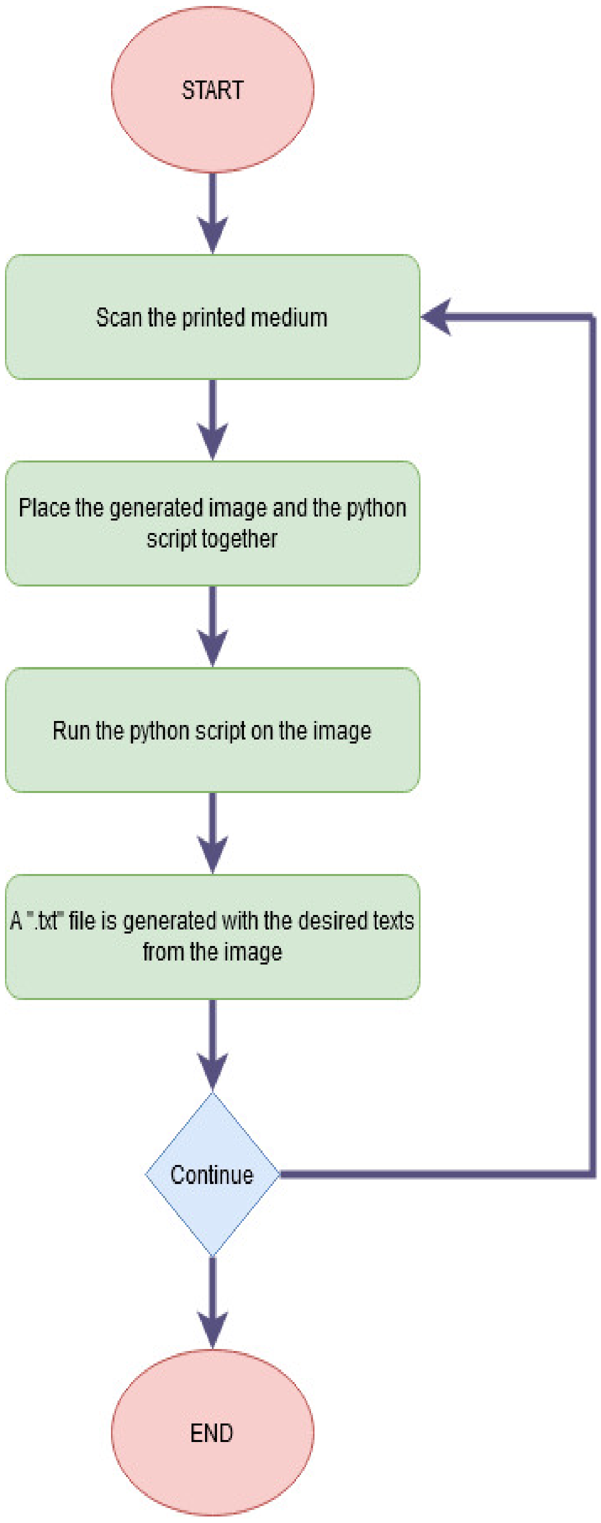

START
Scan the printed medium
Place the generated image and the python script together
Run the python script on the image
A ".txt" file is generated with the desired texts from the image
Continue
END

Figure 5.2: Flowchart of System Use

Chapter 6

System Implementation

The steps for developing and implementing a Bangla OCR application are listed below:

- Preparing the training data/dataset of the Bangla font.
- Training the OCR engine for the Bangla font.
- Performing recognition using the Tesseract engine.

6.1 Required Tools and Programs

Tools required to implement the OCR application are listed below:

- Tesseract 5.0
- jTessEditorFX 2.3.1
- Visual Studio Code
- Python

6.2 Preparing Training Data

The very first thing we need to do to train the engine is to prepare a dataset. This dataset needs to contain all combinations of the vowels, consonants and modifiers in the Bangla language. We need to determine the full character set that needs to be trained along with all its combinations. We need to make a text file which contains all these combinations of characters, where a character sample of at least 10 is good, 5 if the characters are rare. The more frequent characters that appear in the Bangla language should have a sample amount of 20 [8]. We also need to make sure that the dataset is encoded in UTF-8, otherwise the ".txt" to ".tif" converter will be unable to do its job. This paper has prepared and used a dataset that has 3200 units of data, where each unit is a character or a permutation of a character.[8]

The dataset generation can be done in 2 ways for the modifiers of the Bangla language. We have considered both and went with the combining the 2 approaches, as we felt it would give a better recognition rate for the engine.

6.2.1 Dataset with dependent modifiers separate from the basic characters

A word in Bangla consists of some modifiers (vowels and consonants) wrapped around a basic character, according to the grammatical structure of the language. In almost all of the cases, the modifiers and the basic characters overlap with each other [8]. In this case, it is very difficult for Tesseract to make a horizontal or vertical boundary separately for the characters and modifiers. This is no problem for the English language, where the characters do not overlap with each other at all and there's no modifiers. Tesseract can, with some success, recognize certain characters or modifiers that it can separate and that has less overlapping [8]. However, in the rest of the many cases it fails to recognize them as Tesseract cannot separate the modifiers simply using a vertical box or line and thus fails to identify the bounding box between the character and the modifier.[9]

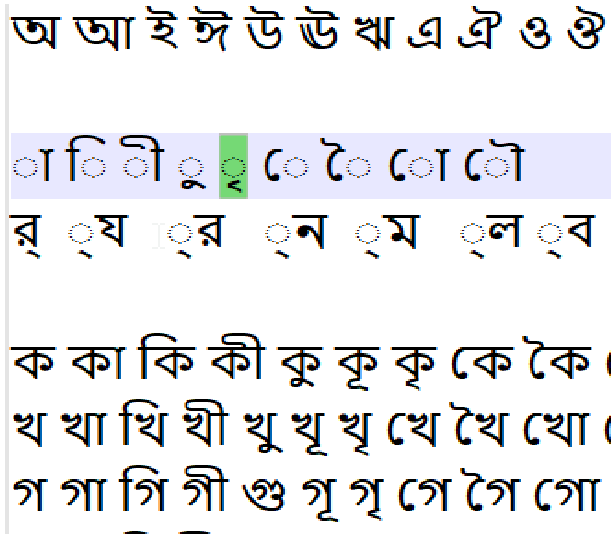
6.2.2 Dataset with dependent modifiers combined with the basic characters

To overcome the limitations presented in 6.2.1, a dataset that contains all the combinations of the characters and any modifiers is used. Thus, the engine can now be trained for all possible combinations, instead of separately where it might fail to recognize a compound character.[9]

Therefore, the dataset will contain all the listed combinations:

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

Combining these 2 approaches of preparing the dataset, we have gathered a dataset consisting of 3200 units and trained for the Siyam Rupali, Sagar and Akaash fonts.



অ আ ই ঈ উ ঊ ঋ এ ঐ ও ঔ

◌া ◌ি ◌ী ◌ু ◌ূ ◌ে ◌ৈ ◌ো ◌ৌ

র্ ◌্য ◌্র ◌্ন ◌্ম ◌্ল ◌্ব

ক কা কি কী কু কূ কৃ কে কৈ

খ খা খি খী খু খূ খৃ খে খৈ খো

গ গা গি গী গু গূ গৃ গে গৈ গো

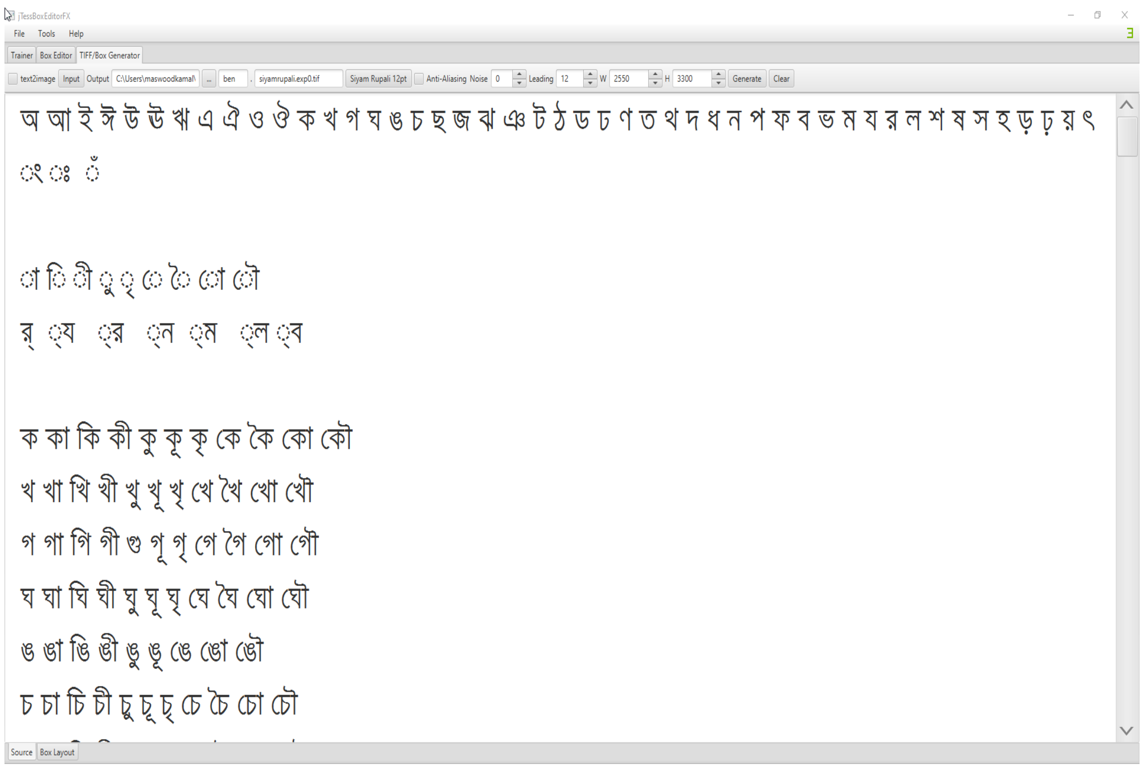
Figure 6.1: Dataset

6.3 Training the OCR for a Bangla Font

The training process has been described and illustrated in the following subsections.

6.3.1 Generate Training Images

After we have made the dataset text file containing all the possible combinations of the language in the desired font, we will have to convert this text file into a ".tif" formatted image. We do this conversion by using a tool called jTessEditorFX. This software is made using java and requires Java Runtime Environment (jre). From past papers, it was suggested as a good measure to artificially add some noise to the TIF image for the OCR to work better on the scanned images from books or newspapers.



text2image Input Output C:\Users\maswoodkamal\ ... ben . siyamrupali.exp0.tif Siyam Rupali 12pt Anti-Aliasing Noise 0 Leading 12 W 2550 H 3300 Generate Clear

অ আ ই ঈ উ ঊ ঋ এ ঐ ও ঔ ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড ঢ ণ ত থ দ ধ ন প ফ ব ভ ম য র ল শ ষ স হ ড় ঢ় য় ৎ

◌ং ◌ঃ ◌ঁ

◌া ◌ি ◌ী ◌ু ◌ূ ◌ৃ ◌ে ◌ৈ ◌ো ◌ৌ

র্ ◌্য ◌্র ◌্ন ◌্ম ◌্ল ◌্ব

ক কা কি কী কু কূ কৃ কে কৈ কো কৌ

খ খা খি খী খু খূ খৃ খে খৈ খো খৌ

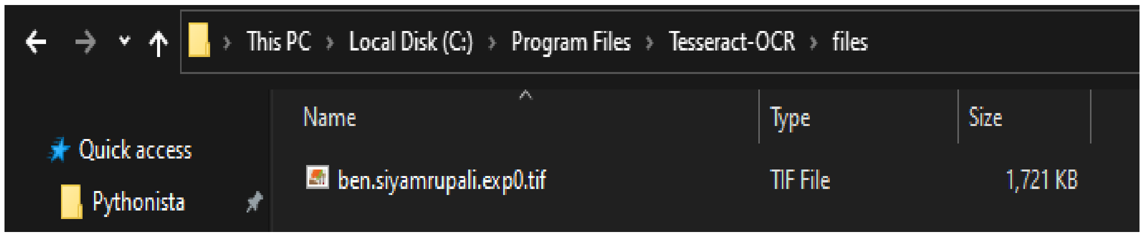
গ গা গি গী গু গূ গৃ গে গৈ গো গৌ

ঘ ঘা ঘি ঘী ঘু ঘূ ঘৃ ঘে ঘৈ ঘো ঘৌ

ঙ ঙা ঙি ঙী ঙু ঙূ ঙে ঙো ঙৌ

চ চা চি চী চু চূ চৃ চে চৈ চো চৌ

Figure 6.2: Generating TIF File in jTessEditor


This PC › Local Disk (C:) › Program Files › Tesseract-OCR › files
Quick access
Pythonista

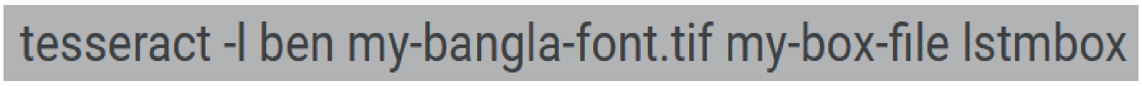
Name	Type	Size
ben.siyamrupali.exp0.tif	TIF File	1,721 KB

Figure 6.3: Generated TIF Image

6.3.2 Generate Box Files

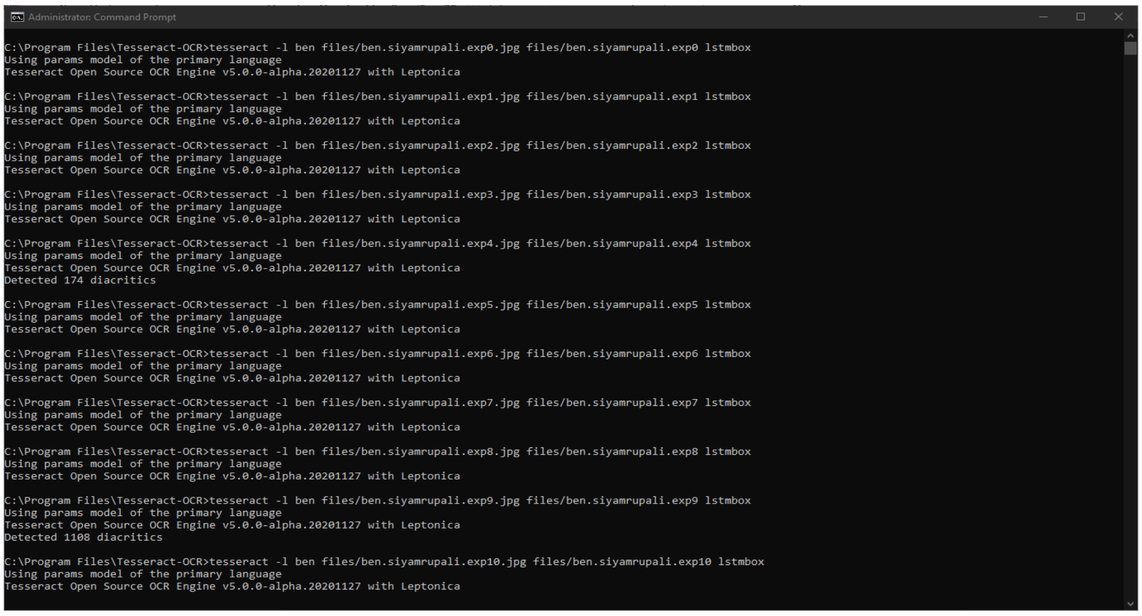
Training the OCR engine, requires us to create box files from the respective TIF images. A box-file is a UTF-8 encoded, plain text file, which has the coordinates of a given text or a character in the image and the needed information of the bounding box of the character or the line [21]. Since this is a LSTM system, the coordinates of an entire line is considered instead of an individual character in the image. The box file generated will be much different from the ones that may use the coordinates of the characters instead [21]. Tesseract itself has some builtin tools which helps us in generating a box file for a TIF image. We call a command from the Tesseract directory to generate our box files for the images. Sometimes, the box files might contain some errors or anomalies, and this requires us to manually edit the box files using jTessEditorFX. jTessEditorFX, not only helps us convert images, but also helps when we need to edit the boxes in the box files, like merging boxes for vowel/consonant with modifiers and compound characters, change the width and height of the box if it fails to capture a line or a character in a bounding box.

For our purposes, we first converted the whole “.tif” image into separate “.jpg” images. We than ran the below command to generate box files for each of those “.jpg” images which are really just the different pages of the “.tif” image.



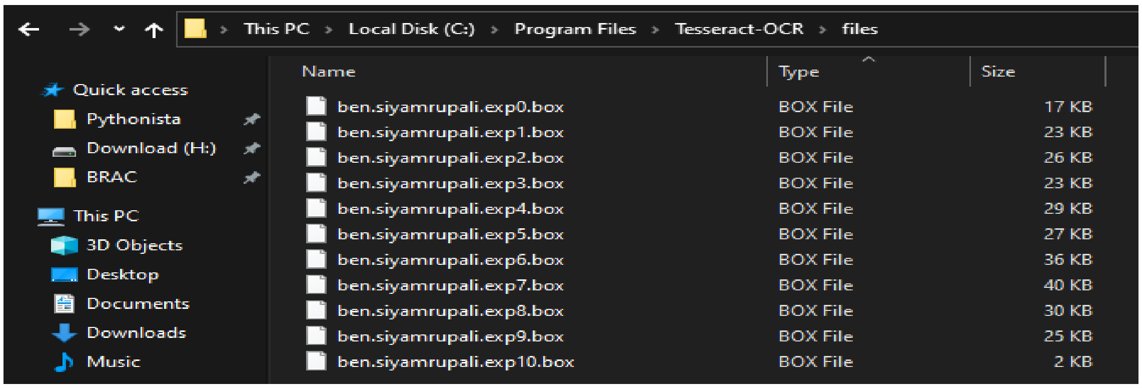
```
tesseract -l ben my-bangla-font.tif my-box-file lstmbox
```

Figure 6.4: Command for generating Box File



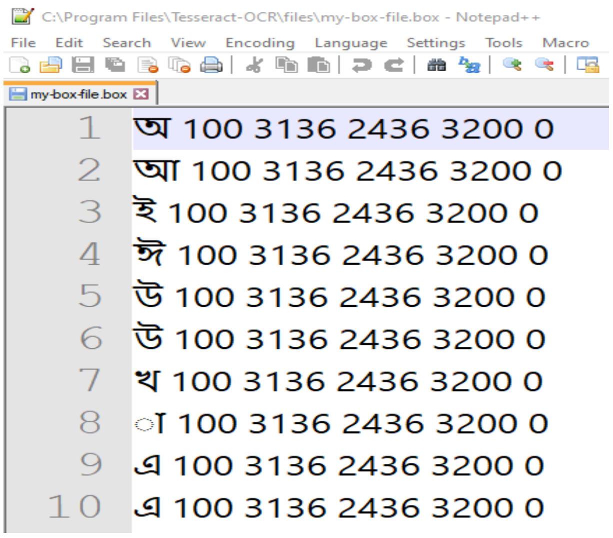
```
C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp0.jpg files/ben.siyamrupali.exp0 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp1.jpg files/ben.siyamrupali.exp1 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp2.jpg files/ben.siyamrupali.exp2 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp3.jpg files/ben.siyamrupali.exp3 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp4.jpg files/ben.siyamrupali.exp4 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica
Detected 174 diacritics

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp5.jpg files/ben.siyamrupali.exp5 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp6.jpg files/ben.siyamrupali.exp6 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp7.jpg files/ben.siyamrupali.exp7 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp8.jpg files/ben.siyamrupali.exp8 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp9.jpg files/ben.siyamrupali.exp9 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica
Detected 1108 diacritics

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp10.jpg files/ben.siyamrupali.exp10 lstmbox
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica
```

Figure 6.5: Generating Box Files



Name	Type	Size
ben.siyamrupali.exp0.box	BOX File	17 KB
ben.siyamrupali.exp1.box	BOX File	23 KB
ben.siyamrupali.exp2.box	BOX File	26 KB
ben.siyamrupali.exp3.box	BOX File	23 KB
ben.siyamrupali.exp4.box	BOX File	29 KB
ben.siyamrupali.exp5.box	BOX File	27 KB
ben.siyamrupali.exp6.box	BOX File	36 KB
ben.siyamrupali.exp7.box	BOX File	40 KB
ben.siyamrupali.exp8.box	BOX File	30 KB
ben.siyamrupali.exp9.box	BOX File	25 KB
ben.siyamrupali.exp10.box	BOX File	2 KB

Figure 6.6: Generated Box Files


C:\Program Files\Tesseract-OCR\files\my-box-file.box - Notepad++
File Edit Search View Encoding Language Settings Tools Macro
my-box-file.box

```
অ 100 3136 2436 3200 0
আ 100 3136 2436 3200 0
ই 100 3136 2436 3200 0
ঈ 100 3136 2436 3200 0
উ 100 3136 2436 3200 0
উ 100 3136 2436 3200 0
ঋ 100 3136 2436 3200 0
◌া 100 3136 2436 3200 0
এ 100 3136 2436 3200 0
এ 100 3136 2436 3200 0
```

Figure 6.7: Box File

<character> <LeftBottom x1> <LeftBottom y1> <RightTop x2> <RightTop y2> <tiff image page number>

Figure 6.8: Format of generated Box File

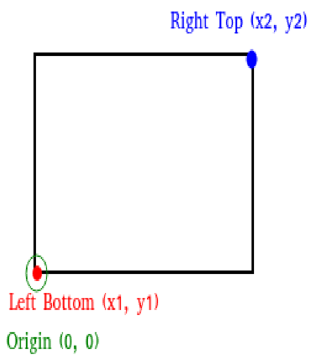
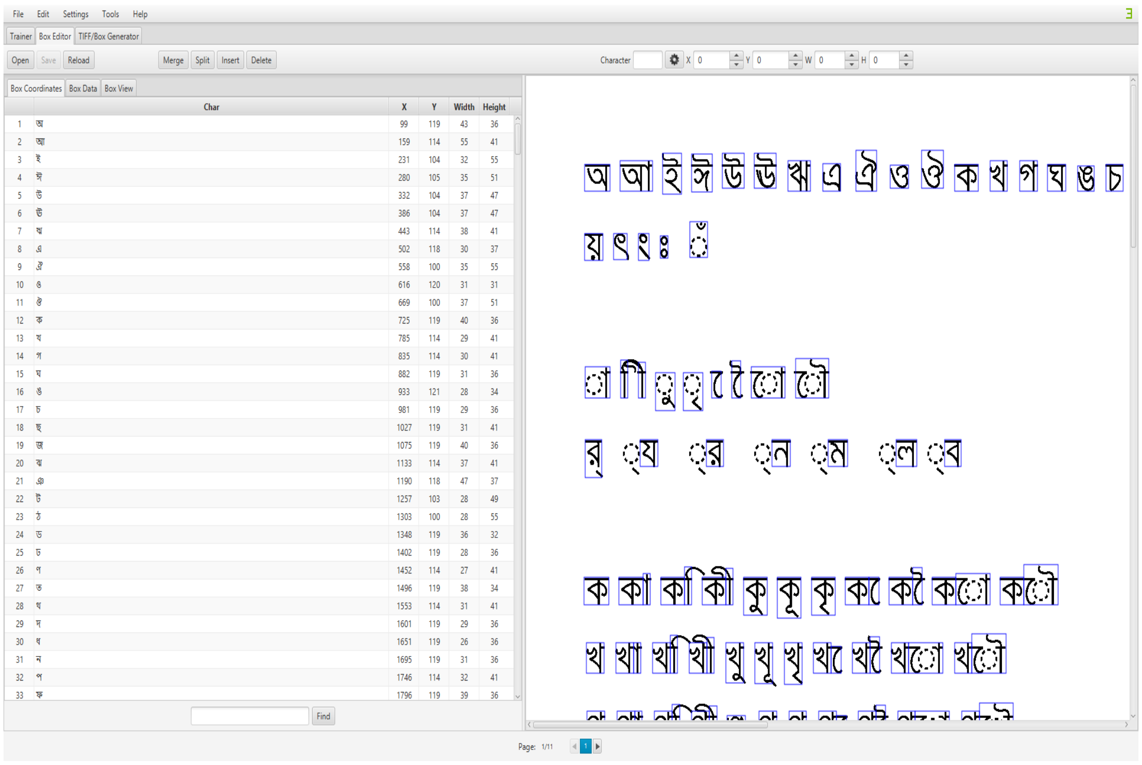


Figure 6.9: Box Coordinate System

6.3.3 Editing Box File

After the box files have been generated, we can open it in jTessEditorFX. We will see a whole bunch of bounding boxes, around our character units. If everything is bounded by boxes, then it means that we do not need to edit anything in the file and can move on to the next step. Alas, reality is harsh and sometimes boxes might simply not appear around some characters, some forms vowels are also treated separately and we would need to merge or split them depending on our character needs. jTessEditorFX gives us a number of options to do these various tasks and we would need to do this task manually for every character which might be special or for its alternate form.

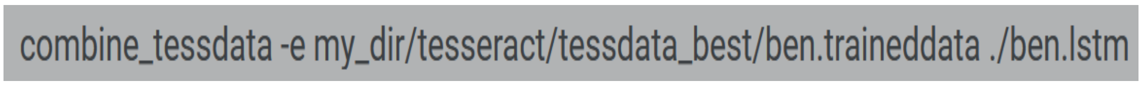


	Char	X	Y	Width	Height
1	অ	99	119	43	36
2	আ	159	114	55	41
3	ই	231	104	32	55
4	ঈ	280	105	35	51
5	উ	332	104	37	47
6	ঊ	386	104	37	47
7	ঋ	443	114	38	41
8	এ	502	118	30	37
9	ঐ	558	100	35	55
10	ও	616	120	31	31
11	ঔ	669	100	37	51
12	ক	725	119	40	36
13	খ	785	114	29	41
14	গ	835	114	30	41
15	ঘ	882	119	31	36
16	ঙ	933	121	28	34
17	চ	981	119	29	36
18	ছ	1027	119	31	41
19	জ	1075	119	40	36
20	ঝ	1133	114	37	41
21	ঞ	1190	118	47	37
22	ট	1257	103	28	49
23	ঠ	1303	100	28	55
24	ড	1348	119	36	32
25	ঢ	1402	119	28	36
26	ণ	1452	114	27	41
27	ত	1496	119	38	34
28	থ	1553	114	31	41
29	দ	1601	119	29	36
30	ধ	1651	119	26	36
31	ন	1695	119	31	36
32	প	1746	114	32	41
33	ফ	1796	119	39	36

Figure 6.10: Editing Bounding Boxes

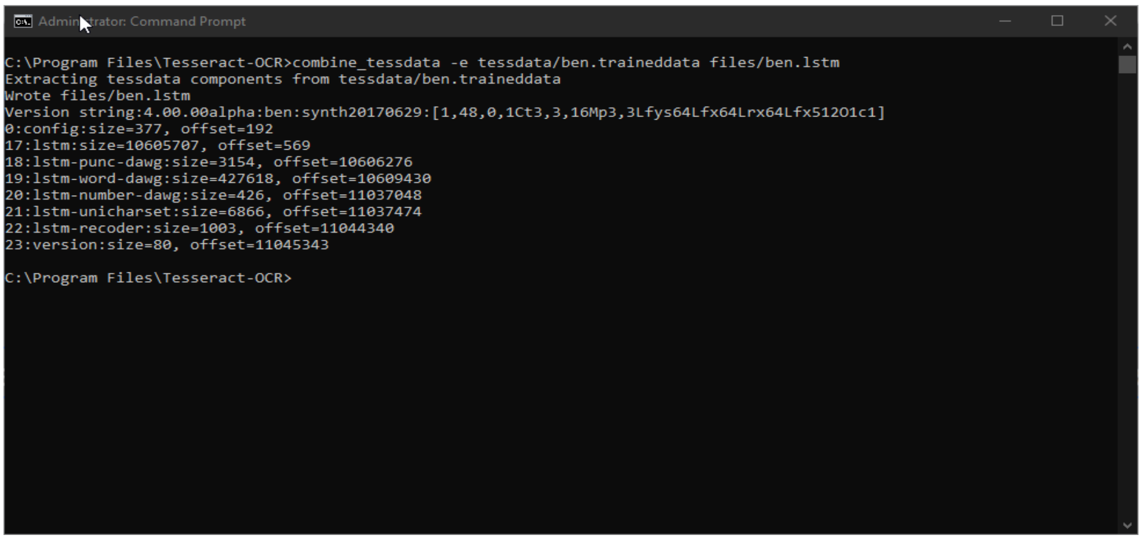
6.3.4 Extracting the Recognition Model

We need to extract the recognition model from an existing ".traineddata" file. The output of this would be an ".lstm" file [18]. This paper uses and extracts the recognition model from the github repo tesseract-ocr/tessdata_best as this paper was used using Tesseract 5 which has a LSTM RNN.



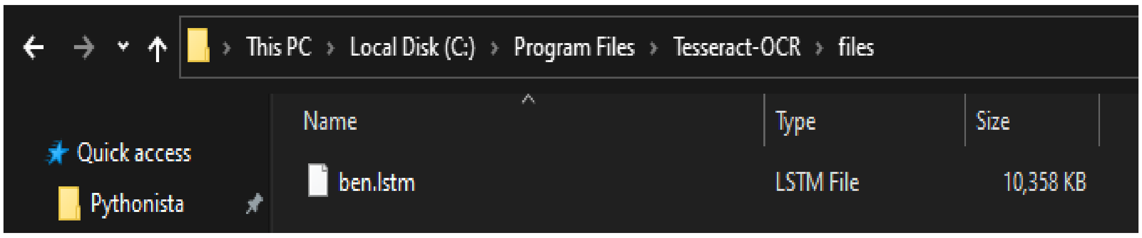
```
combine_tessdata -e my_dir/tesseract/tessdata_best/ben.traineddata ./ben.lstm
```

Figure 6.11: Command for Extracting Recognition Model



```
C:\Program Files\Tesseract-OCR>combine_tessdata -e tessdata/ben.traineddata files/ben.lstm
Extracting tessdata components from tessdata/ben.traineddata
Wrote files/ben.lstm
Version string:4.00.00alpha:ben:synth20170629:[1,48,0,1Ct3,3,16Mp3,3Lfys64Lfx64Lrx64Lfx512O1c1]
0:config:size=377, offset=192
17:lstm:size=10605707, offset=569
18:lstm-punc-dawg:size=3154, offset=10606276
19:lstm-word-dawg:size=427618, offset=10609430
20:lstm-number-dawg:size=426, offset=11037048
21:lstm-unicharset:size=6866, offset=11037474
22:lstm-recoder:size=1003, offset=11044340
23:version:size=80, offset=11045343

C:\Program Files\Tesseract-OCR>
```

Figure 6.12: Generating LSTM File

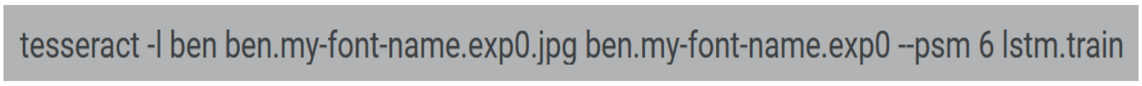

This PC › Local Disk (C:) › Program Files › Tesseract-OCR › files
Quick access
Pythonista

Name	Type	Size
ben.lstm	LSTM File	10,358 KB

Figure 6.13: Generated LSTM File

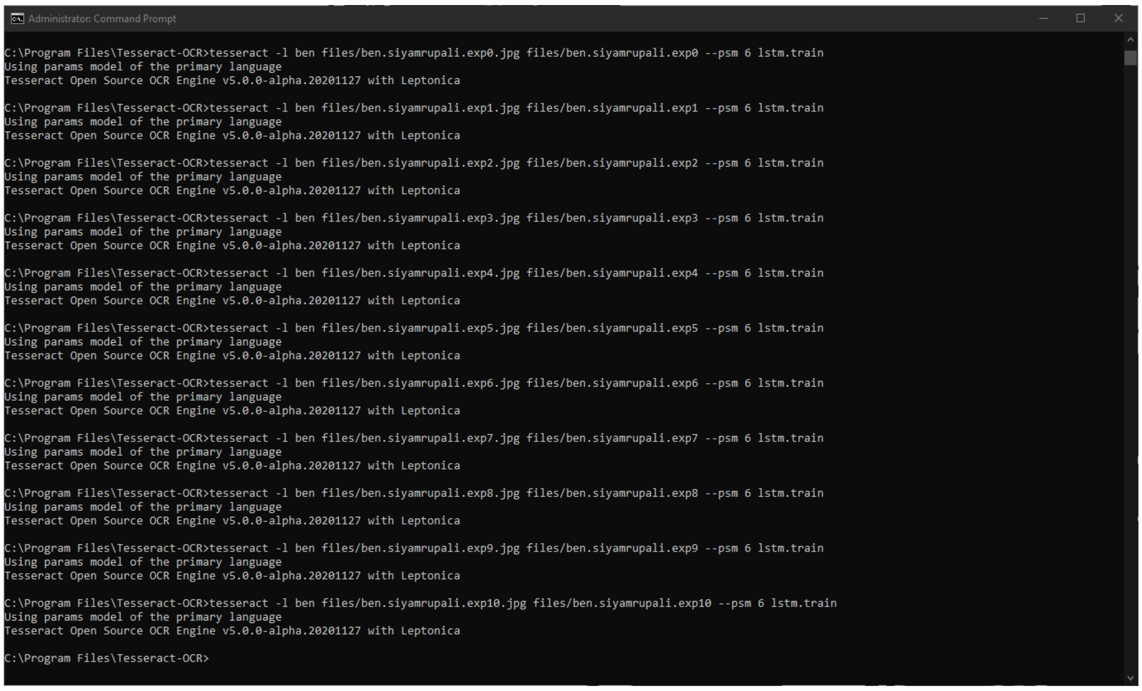
6.3.5 Creating LSTMF Files

We would also need a lstmf binary file from a ".jpg" image and its box file. The training data is provided via this ".lstmf" file, which is a serialized DocumentData [21]. It contains an image and its respective UTF-8 text transcription [21]. The process here is similar to the way the older engines used to create ".tr" files. We create multiple ".lstmf" files from all our JPG/box pairs and make a list in a text file called "ben.training_files.txt" [18], where all the ".lstmf" files are listed with their full paths.



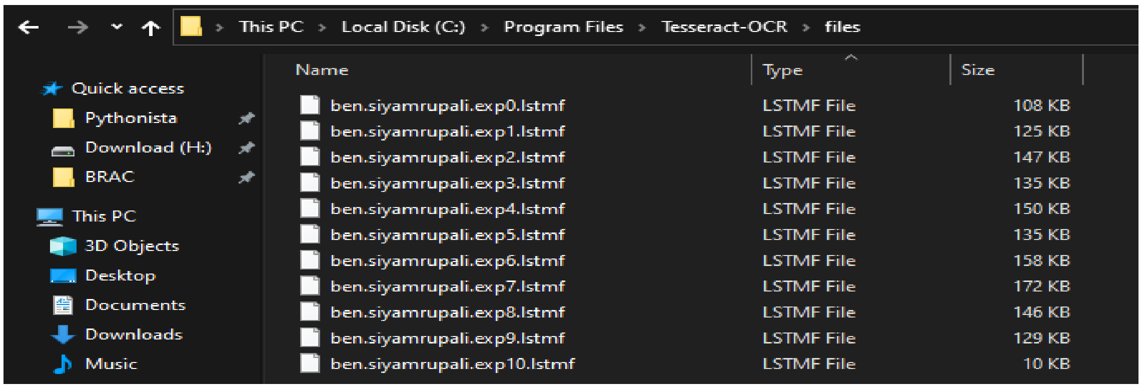
```
tesseract -l ben ben.my-font-name.exp0.jpg ben.my-font-name.exp0 --psm 6 lstm.train
```

Figure 6.14: Command for creating LSTMF Files



```
C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp0.jpg files/ben.siyamrupali.exp0 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp1.jpg files/ben.siyamrupali.exp1 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp2.jpg files/ben.siyamrupali.exp2 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp3.jpg files/ben.siyamrupali.exp3 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp4.jpg files/ben.siyamrupali.exp4 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp5.jpg files/ben.siyamrupali.exp5 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp6.jpg files/ben.siyamrupali.exp6 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp7.jpg files/ben.siyamrupali.exp7 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp8.jpg files/ben.siyamrupali.exp8 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp9.jpg files/ben.siyamrupali.exp9 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>tesseract -l ben files/ben.siyamrupali.exp10.jpg files/ben.siyamrupali.exp10 --psm 6 lstm.train
Using params model of the primary language
Tesseract Open Source OCR Engine v5.0.0-alpha.20201127 with Leptonica

C:\Program Files\Tesseract-OCR>
```

Figure 6.15: Generating LSTMF Files



Name	Type	Size
ben.siyamrupali.exp0.lstmf	LSTMF File	108 KB
ben.siyamrupali.exp1.lstmf	LSTMF File	125 KB
ben.siyamrupali.exp2.lstmf	LSTMF File	147 KB
ben.siyamrupali.exp3.lstmf	LSTMF File	135 KB
ben.siyamrupali.exp4.lstmf	LSTMF File	150 KB
ben.siyamrupali.exp5.lstmf	LSTMF File	135 KB
ben.siyamrupali.exp6.lstmf	LSTMF File	158 KB
ben.siyamrupali.exp7.lstmf	LSTMF File	172 KB
ben.siyamrupali.exp8.lstmf	LSTMF File	146 KB
ben.siyamrupali.exp9.lstmf	LSTMF File	129 KB
ben.siyamrupali.exp10.lstmf	LSTMF File	10 KB

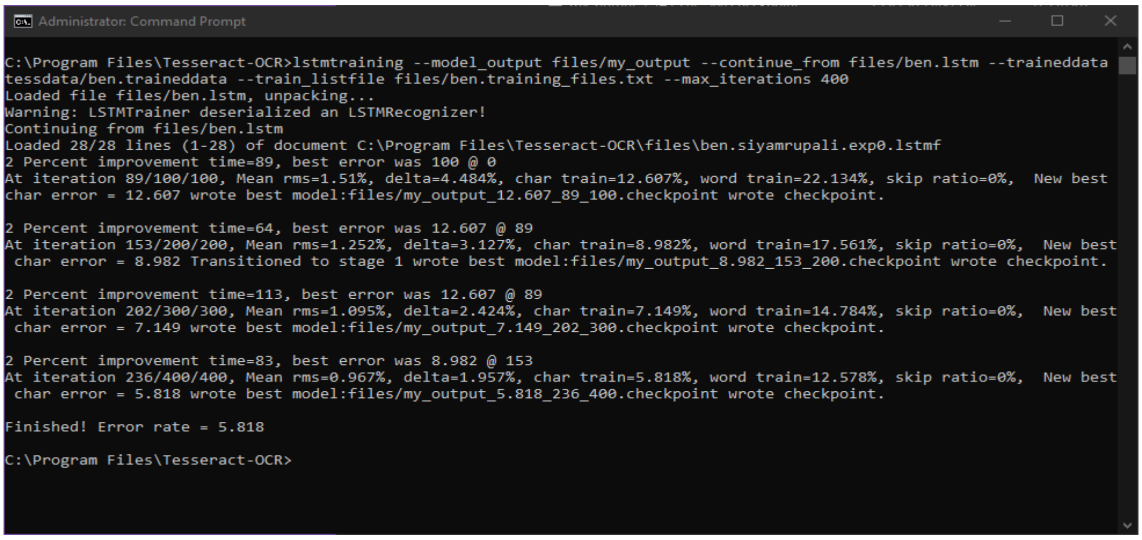
Figure 6.16: Generated LSTMF Files

6.3.6 Training the Engine

After all the ".lstmf" files have been created, we will proceed to finally training the engine for our font. A standard behaviour for any neural network, such as the one in Tesseract, is to write checkpoint files during its training. The cause of making these checkpoint files is so that the training can be stopped and continued when we desire. The Tesseract trainer also makes various checkpoint files for when the training has hit a new best record or achievement in training for our font [21]. The "--continue_from" parameter in the below command, also lets us modify and retrain the network or just fine tune specific parts of the language, by letting the trainer "continue from" a naked ".lstm" file like we did below or even from a checkpoint file. The "--model_output" parameter lets the trainer periodically save checkpoints. Thus it is possible to stop the training midway and restart it again from that checkpoint.[21]

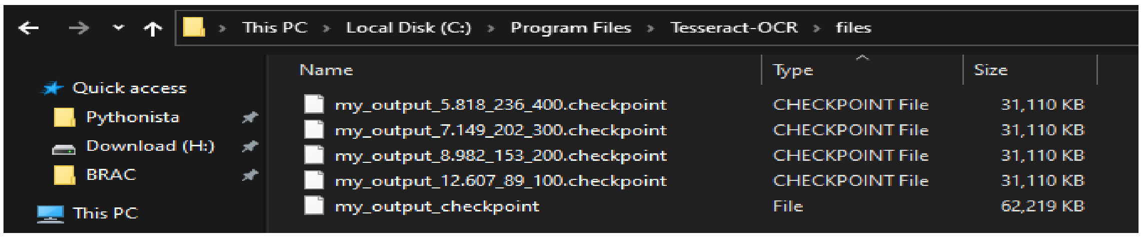
```
lstmtraining --model_output ./my_output --continue_from ./ben.lstm --traineddata my_dir/tesseract/
tessdata_best/ben.traineddata --train_listfile ./ben.training_files.txt --max_iterations 400
```

Figure 6.17: Command for starting the Training of the OCR



```
C:\Program Files\Tesseract-OCR>lstmtraining --model_output files/my_output --continue_from files/ben.lstm --traineddata
tessdata/ben.traineddata --train_listfile files/ben.training_files.txt --max_iterations 400
Loaded file files/ben.lstm, unpacking...
Warning: LSTMTrainer deserialized an LSTMRecognizer!
Continuing from files/ben.lstm
Loaded 28/28 lines (1-28) of document C:\Program Files\Tesseract-OCR\files\ben.siyamrupali.exp0.lstmf
2 Percent improvement time=89, best error was 100 @ 0
At iteration 89/100/100, Mean rms=1.51%, delta=4.484%, char train=12.607%, word train=22.134%, skip ratio=0%,  New best
char error = 12.607 wrote best model:files/my_output_12.607_89_100.checkpoint wrote checkpoint.

2 Percent improvement time=64, best error was 12.607 @ 89
At iteration 153/200/200, Mean rms=1.252%, delta=3.127%, char train=8.982%, word train=17.561%, skip ratio=0%,  New best
 char error = 8.982 Transitioned to stage 1 wrote best model:files/my_output_8.982_153_200.checkpoint wrote checkpoint.

2 Percent improvement time=113, best error was 12.607 @ 89
At iteration 202/300/300, Mean rms=1.095%, delta=2.424%, char train=7.149%, word train=14.784%, skip ratio=0%,  New best
 char error = 7.149 wrote best model:files/my_output_7.149_202_300.checkpoint wrote checkpoint.

2 Percent improvement time=83, best error was 8.982 @ 153
At iteration 236/400/400, Mean rms=0.967%, delta=1.957%, char train=5.818%, word train=12.578%, skip ratio=0%,  New best
 char error = 5.818 wrote best model:files/my_output_5.818_236_400.checkpoint wrote checkpoint.

Finished! Error rate = 5.818

C:\Program Files\Tesseract-OCR>
```

Figure 6.18: Training the OCR Engine



This PC > Local Disk (C:) > Program Files > Tesseract-OCR > files
Quick access
Pythonista
Download (H:)
BRAC
This PC

Name	Type	Size
my_output_5.818_236_400.checkpoint	CHECKPOINT File	31,110 KB
my_output_7.149_202_300.checkpoint	CHECKPOINT File	31,110 KB
my_output_8.982_153_200.checkpoint	CHECKPOINT File	31,110 KB
my_output_12.607_89_100.checkpoint	CHECKPOINT File	31,110 KB
my_output_checkpoint	File	62,219 KB

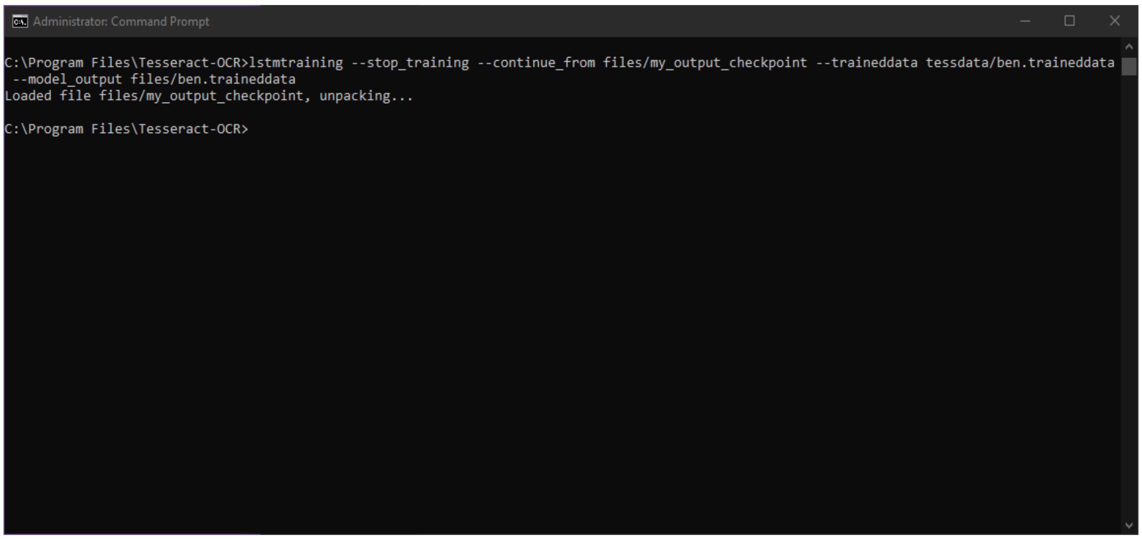
Figure 6.19: Generated Checkpoint Files

6.3.7 Combining

Once the training process has run its course, it is time for us to generate the new ".traineddata" file by combining it with our checkpoint file from our previous step and our existing ".traineddata" file. By using the "--stop_training" parameter in our below code, we're telling the engine to use our checkpoint file and convert it to a fully fledged ".traineddata" file ready for recognition. The "--model_output" parameter tells the engine to save the file in the desired basename [21]. This new ".traineddata" file is what we are going to use from now on as our default traineddata file, since this is optimized and trained for our font and as well as has the support to recognize the previous fonts from the old traineddata file.

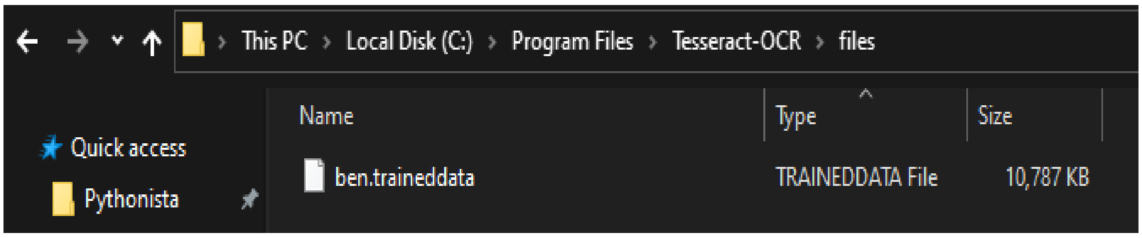
```
lstmtraining --stop_training --continue_from ./my_output_checkpoint --traineddata my_dir/tesseract/
tessdata_best/ben.traineddata --model_output my_dir/tessdata_best/ben.traineddata
```

Figure 6.20: Command for combining the TRAINEDDATA Files



```
C:\Program Files\Tesseract-OCR>lstmtraining --stop_training --continue_from files/my_output_checkpoint --traineddata tessdata/ben.traineddata
 --model_output files/ben.traineddata
Loaded file files/my_output_checkpoint, unpacking...

C:\Program Files\Tesseract-OCR>
```

Figure 6.21: Generating new TRAINEDDATA File


This PC › Local Disk (C:) › Program Files › Tesseract-OCR › files
Quick access
Pythonista

Name	Type	Size
ben.traineddata	TRAINEDDATA File	10,787 KB

Figure 6.22: Generated TRAINEDDATA File

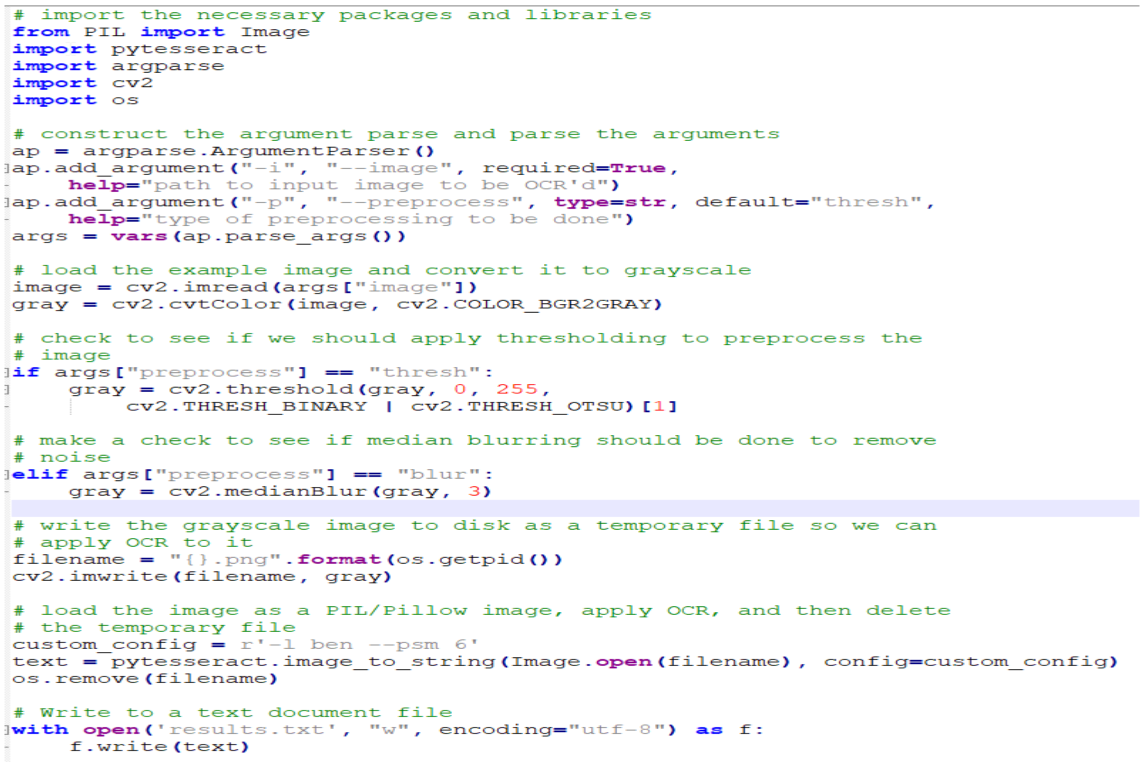
6.4 Performing Recognition with the Tesseract Engine

After we have trained the engine and generated our new ".traineddata" file, it is now time to use the engine for recognition of scanned images from printed mediums like, newspapers, books, journals etc. For our convenience and simplicity, we have used the Python programming language to write a script that would do our recognition for us and give a text file as an output which would contain the text that we are supposed to extract from our images. We could have run the recognition using Tesseract and the command line interface provided by Windows but then we would have to write a big one-liner command, everytime for every image, where we have to set our parameters again. The easy part of using a python script is that we can just set the parameters we want as a default inside the script and need to only mention the image directory and the script's name in the command line interface to run the script ready with all the arguments.

We have to do a bit of preprocessing too to run the image for recognition. This is where using the python script shines more. The script itself is going to preprocess the image for us before the recognition process starts, so that we don't have to do it separately for each image, and give us our desired output.

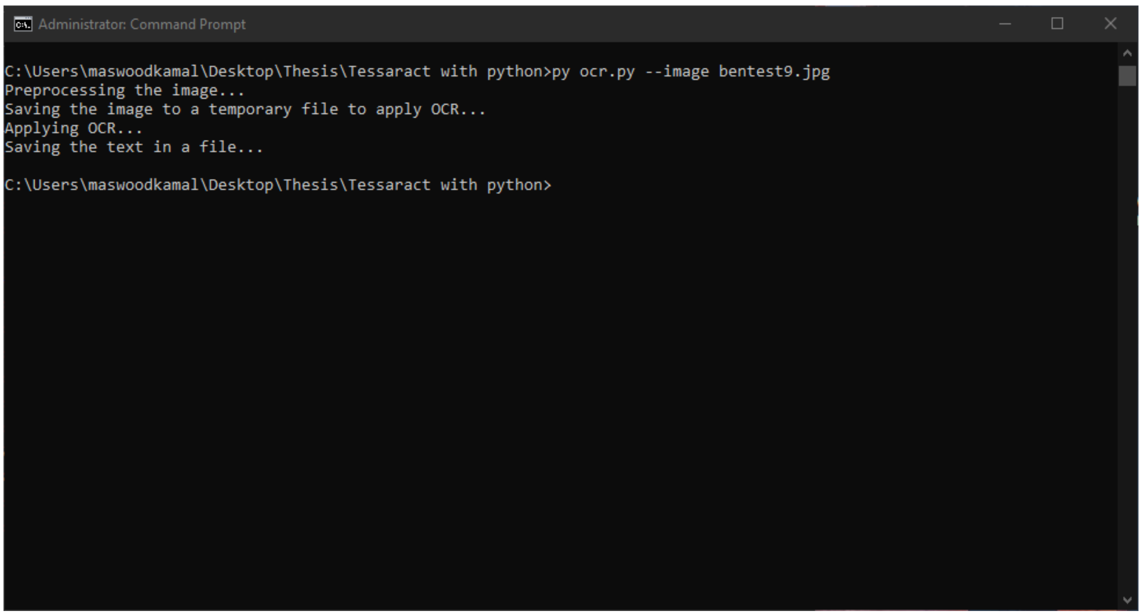
6.4.1 Overview of the Recognition Script

The script first imports the necessary libraries and tools using a new "imports". Then the script will require 1 argument by default, which is the image location and name and the second argument it asks for in the command prompt before it runs the script is the pre-processing method, but this is optional.
After we have provided the command to run the script, it will first read the image and convert it to grayscale image. After grayscaling the image, the next step that the image goes through is the binarization method. Here, the script uses Otsu's Thresholding method to binarize the image, which is one of the best, if not the best method tailored for our needs. We can also use a Blur method here for some noisy images containing blur in the image. The binarized image is then saved temporarily in the run directory for the next step. Once the binarized image is generated and saved for the time being, the script itself runs the command used for recognition in the command prompt which uses the Tesseract engine to recognize the texts in the image and extract them. Afterwards, when the extraction is fully complete on a particular image, the extracted text is saved in a text file in the current directory.



```python
# import the necessary packages and libraries
from PIL import Image
import pytesseract
import argparse
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image to be OCR'd")
ap.add_argument("-p", "--preprocess", type=str, default="thresh",
    help="type of preprocessing to be done")
args = vars(ap.parse_args())

# load the example image and convert it to grayscale
image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# check to see if we should apply thresholding to preprocess the
# image
if args["preprocess"] == "thresh":
    gray = cv2.threshold(gray, 0, 255,
        cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

# make a check to see if median blurring should be done to remove
# noise
elif args["preprocess"] == "blur":
    gray = cv2.medianBlur(gray, 3)

# write the grayscale image to disk as a temporary file so we can
# apply OCR to it
filename = "{}.png".format(os.getpid())
cv2.imwrite(filename, gray)

# load the image as a PIL/Pillow image, apply OCR, and then delete
# the temporary file
custom_config = r'-l ben --psm 6'
text = pytesseract.image_to_string(Image.open(filename), config=custom_config)
os.remove(filename)

# Write to a text document file
with open('results.txt', "w", encoding="utf-8") as f:
    f.write(text)
```

Figure 6.23: Python Script



```
C:\Users\maswoodkamal\Desktop\Thesis\Tessaract with python>py ocr.py --image bentest9.jpg
Preprocessing the image...
Saving the image to a temporary file to apply OCR...
Applying OCR...
Saving the text in a file...

C:\Users\maswoodkamal\Desktop\Thesis\Tessaract with python>
```

Figure 6.24: Running the Python Script

Chapter 7

Result Analysis

We used Tesseract 5.0 for our entire project. Tesseract 5.0 comes with its own improvements over Tesseract 3.0 and 4.0. We had varying accuracy according to the type of picture that was used for recognition. Figure shows extracted font from a single line of text. The input and output are shown below. A grayscale image produces the best result.

All of our results are based on best case scenarios, we can see accuracy up to a 100% as well. We used a simple formula to calculate the accuracy of our experiments, which is

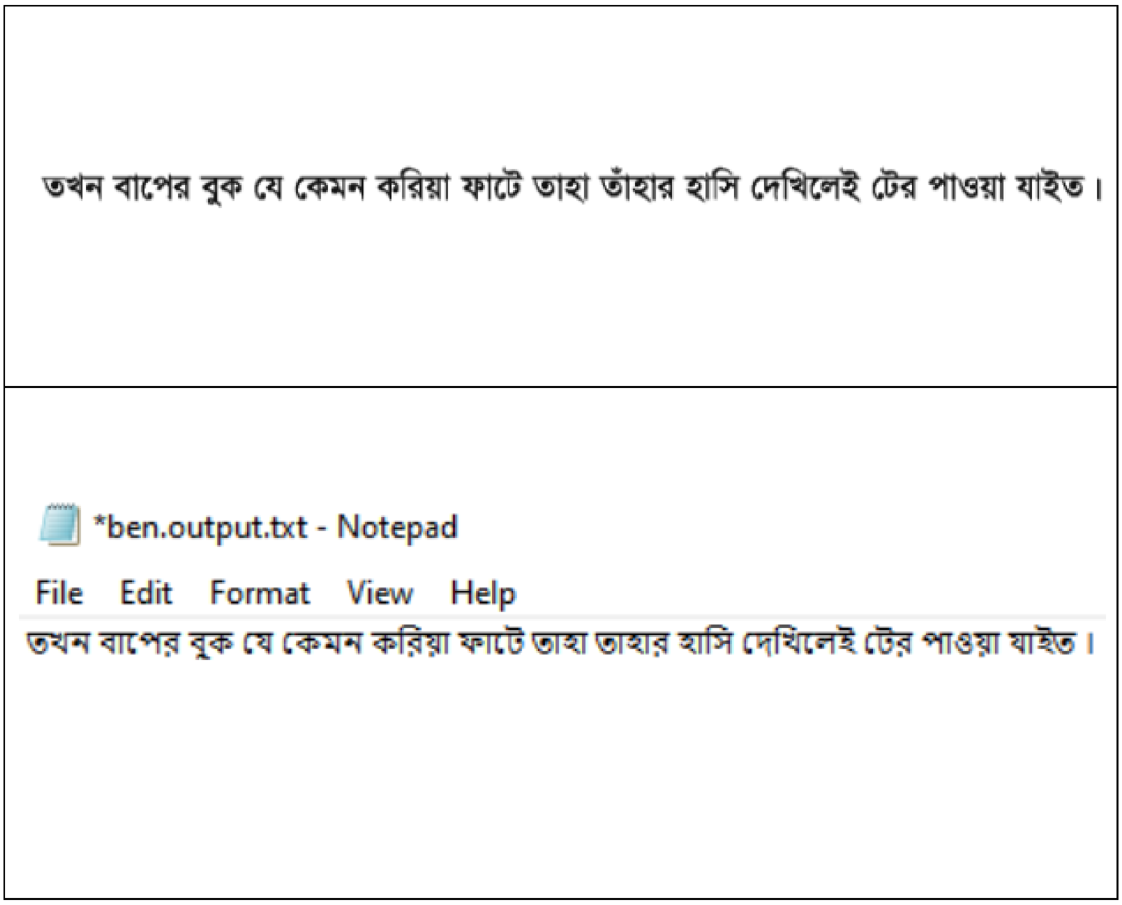
$$\text{Accuracy} = \frac{\text{Total Number of Words - Total Number of Mistakes}}{\text{Total Number of Words}} 100\% \tag{7.1}$$

[19]

We ran an accuracy test on a word based level. In Figure 7.7, we can see that there are 92 words and there is one mistake so the accuracy comes out to be 98.9%. Our average accuracy across several test images and fonts came out to be 95.9%.

7.1 Experiment using Single-Line Text Image

Below is an example of single-line text image.



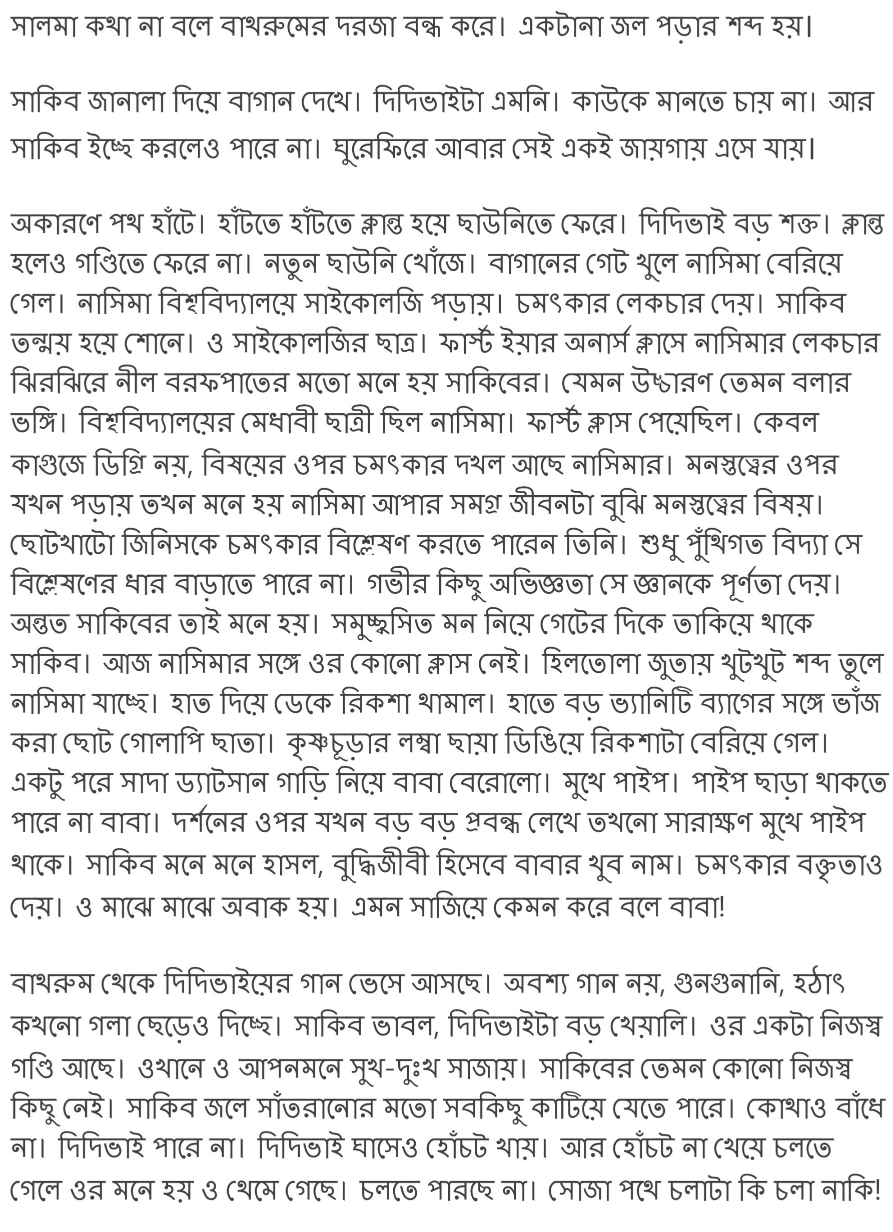
তখন বাপের বুক যে কেমন করিয়া ফাটে তাহা তাঁহার হাসি দেখিলেই টের পাওয়া যাইত।

তখন বাপের বুক যে কেমন করিয়া ফাটে তাহা তাহার হাসি দেখিলেই টের পাওয়া যাইত।

Figure 7.1: Extraction from single line text image

7.2 Experiment using Multi-Line Text Image

Below is a comparison of input and output on multiple line text image from a textbook.



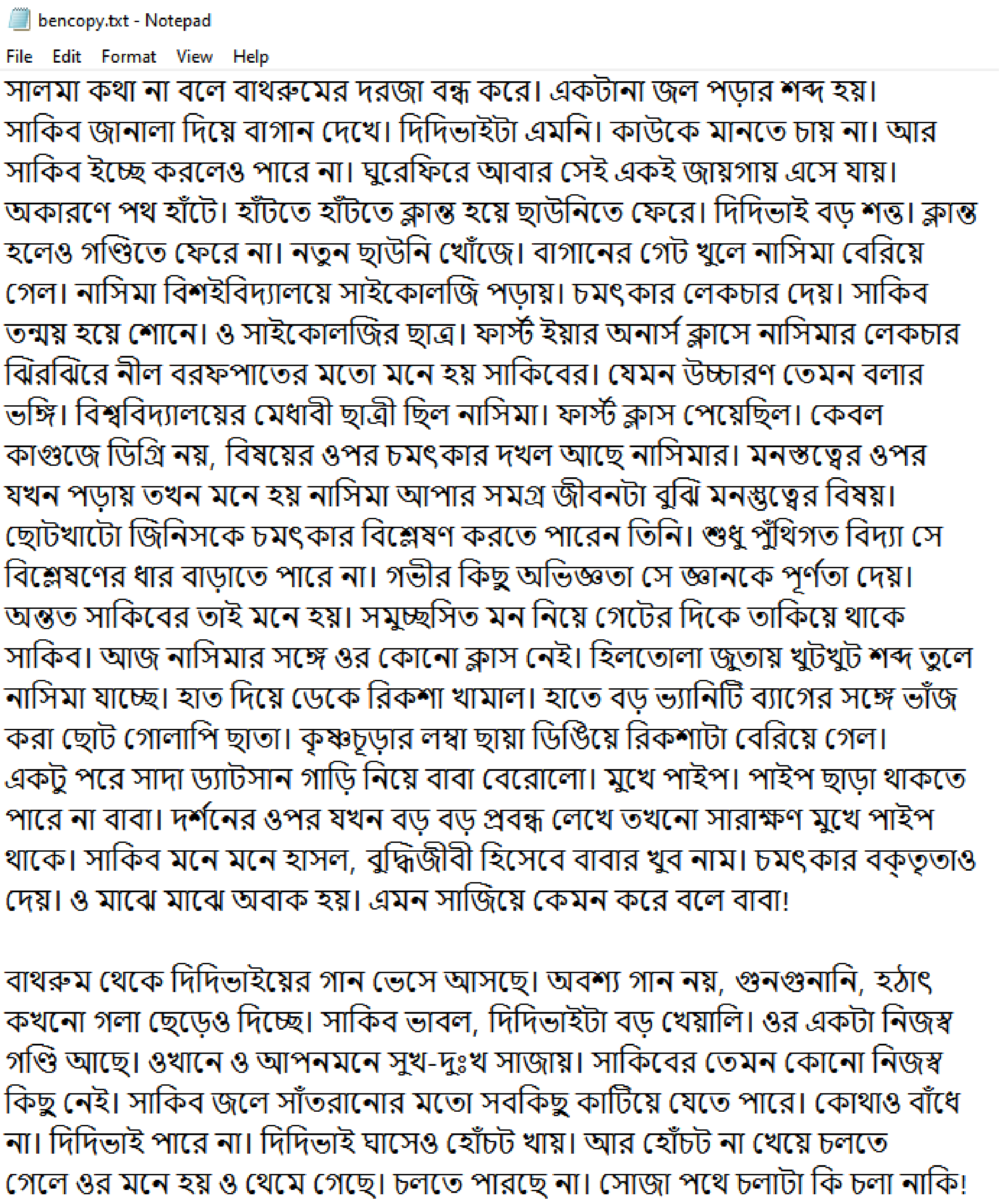
সালমা কথা না বলে বাথরুমের দরজা বন্ধ করে। একটানা জল পড়ার শব্দ হয়।

সাকিব জানালা দিয়ে বাগান দেখে। দিদিভাইটা এমনি। কাউকে মানতে চায় না। আর
সাকিব ইচ্ছে করলেও পারে না। ঘুরেফিরে আবার সেই একই জায়গায় এসে যায়।

অকারণে পথ হাঁটে। হাঁটতে হাঁটতে ক্লান্ত হয়ে ছাউনিতে ফেরে। দিদিভাই বড় শক্ত। ক্লান্ত
হলেও গণ্ডিতে ফেরে না। নতুন ছাউনি খোঁজে। বাগানের গেট খুলে নাসিমা বেরিয়ে
গেল। নাসিমা বিশ্ববিদ্যালয়ে সাইকোলজি পড়ায়। চমৎকার লেকচার দেয়। সাকিব
তন্ময় হয়ে শোনে। ও সাইকোলজির ছাত্র। ফার্স্ট ইয়ার অনার্স ক্লাসে নাসিমার লেকচার
ঝিরঝিরে নীল বরফপাতের মতো মনে হয় সাকিবের। যেমন উচ্চারণ তেমন বলার
ভঙ্গি। বিশ্ববিদ্যালয়ের মেধাবী ছাত্রী ছিল নাসিমা। ফার্স্ট ক্লাস পেয়েছিল। কেবল
কাগুজে ডিগ্রি নয়, বিষয়ের ওপর চমৎকার দখল আছে নাসিমার। মনস্তত্ত্বের ওপর
যখন পড়ায় তখন মনে হয় নাসিমা আপার সমগ্র জীবনটা বুঝি মনস্তত্ত্বের বিষয়।
ছোটখাটো জিনিসকে চমৎকার বিশ্লেষণ করতে পারেন তিনি। শুধু পুঁথিগত বিদ্যা সে
বিশ্লেষণের ধার বাড়াতে পারে না। গভীর কিছু অভিজ্ঞতা সে জ্ঞানকে পূর্ণতা দেয়।
অন্তত সাকিবের তাই মনে হয়। সমুচ্ছ্বসিত মন নিয়ে গেটের দিকে তাকিয়ে থাকে
সাকিব। আজ নাসিমার সঙ্গে ওর কোনো ক্লাস নেই। হিলতোলা জুতায় খুটখুট শব্দ তুলে
নাসিমা যাচ্ছে। হাত দিয়ে ডেকে রিকশা থামাল। হাতে বড় ভ্যানিটি ব্যাগের সঙ্গে ভাঁজ
করা ছোট গোলাপি ছাতা। কৃষ্ণচূড়ার লম্বা ছায়া ডিঙিয়ে রিকশাটা বেরিয়ে গেল।
একটু পরে সাদা ড্যাটসান গাড়ি নিয়ে বাবা বেরোলো। মুখে পাইপ। পাইপ ছাড়া থাকতে
পারে না বাবা। দর্শনের ওপর যখন বড় বড় প্রবন্ধ লেখে তখনো সারাক্ষণ মুখে পাইপ
থাকে। সাকিব মনে মনে হাসল, বুদ্ধিজীবী হিসেবে বাবার খুব নাম। চমৎকার বক্তৃতাও
দেয়। ও মাঝে মাঝে অবাক হয়। এমন সাজিয়ে কেমন করে বলে বাবা!

বাথরুম থেকে দিদিভাইয়ের গান ভেসে আসছে। অবশ্য গান নয়, গুনগুনানি, হঠাৎ
কখনো গলা ছেড়েও দিচ্ছে। সাকিব ভাবল, দিদিভাইটা বড় খেয়ালি। ওর একটা নিজস্ব
গণ্ডি আছে। ওখানে ও আপনমনে সুখ-দুঃখ সাজায়। সাকিবের তেমন কোনো নিজস্ব
কিছু নেই। সাকিব জলে সাঁতরানোর মতো সবকিছু কাটিয়ে যেতে পারে। কোথাও বাঁধে
না। দিদিভাই পারে না। দিদিভাই ঘাসেও হোঁচট খায়। আর হোঁচট না খেয়ে চলতে
গেলে ওর মনে হয় ও থেমে গেছে। চলতে পারছে না। সোজা পথে চলাটা কি চলা নাকি!

Figure 7.2: Printed text as input


bencopy.txt - Notepad

সালমা কথা না বলে বাথরুমের দরজা বন্ধ করে। একটানা জল পড়ার শব্দ হয়।
সাকিব জানালা দিয়ে বাগান দেখে। দিদিভাইটা এমনি। কাউকে মানতে চায় না। আর
সাকিব ইচ্ছে করলেও পারে না। ঘুরেফিরে আবার সেই একই জায়গায় এসে যায়।
অকারণে পথ হাঁটে। হাঁটতে হাঁটতে ক্লান্ত হয়ে ছাউনিতে ফেরে। দিদিভাই বড় শক্ত। ক্লান্ত
হলেও গণ্ডিতে ফেরে না। নতুন ছাউনি খোঁজে। বাগানের গেট খুলে নাসিমা বেরিয়ে
গেল। নাসিমা বিশইবিদ্যালয়ে সাইকোলজি পড়ায়। চমৎকার লেকচার দেয়। সাকিব
তন্ময় হয়ে শোনে। ও সাইকোলজির ছাত্র। ফার্স্ট ইয়ার অনার্স ক্লাসে নাসিমার লেকচার
ঝিরঝিরে নীল বরফপাতের মতো মনে হয় সাকিবের। যেমন উচ্চারণ তেমন বলার
ভঙ্গি। বিশ্ববিদ্যালয়ের মেধাবী ছাত্রী ছিল নাসিমা। ফার্স্ট ক্লাস পেয়েছিল। কেবল
কাগুজে ডিগ্রি নয়, বিষয়ের ওপর চমৎকার দখল আছে নাসিমার। মনস্তত্ত্বের ওপর
যখন পড়ায় তখন মনে হয় নাসিমা আপার সমগ্র জীবনটা বুঝি মনস্তত্ত্বের বিষয়।
ছোটখাটো জিনিসকে চমৎকার বিশ্লেষণ করতে পারেন তিনি। শুধু পুঁথিগত বিদ্যা সে
বিশ্লেষণের ধার বাড়াতে পারে না। গভীর কিছু অভিজ্ঞতা সে জ্ঞানকে পূর্ণতা দেয়।
অন্তত সাকিবের তাই মনে হয়। সমুচ্ছ্বসিত মন নিয়ে গেটের দিকে তাকিয়ে থাকে
সাকিব। আজ নাসিমার সঙ্গে ওর কোনো ক্লাস নেই। হিলতোলা জুতায় খুটখুট শব্দ তুলে
নাসিমা যাচ্ছে। হাত দিয়ে ডেকে রিকশা খামাল। হাতে বড় ভ্যানিটি ব্যাগের সঙ্গে ভাঁজ
করা ছোট গোলাপি ছাতা। কৃষ্ণচূড়ার লম্বা ছায়া ডিঙিয়ে রিকশাটা বেরিয়ে গেল।
একটু পরে সাদা ড্যাটসান গাড়ি নিয়ে বাবা বেরোলো। মুখে পাইপ। পাইপ ছাড়া থাকতে
পারে না বাবা। দর্শনের ওপর যখন বড় বড় প্রবন্ধ লেখে তখনো সারাক্ষণ মুখে পাইপ
থাকে। সাকিব মনে মনে হাসল, বুদ্ধিজীবী হিসেবে বাবার খুব নাম। চমৎকার বক্তৃতাও
দেয়। ও মাঝে মাঝে অবাক হয়। এমন সাজিয়ে কেমন করে বলে বাবা!

বাথরুম থেকে দিদিভাইয়ের গান ভেসে আসছে। অবশ্য গান নয়, গুনগুনানি, হঠাৎ
কখনো গলা ছেড়েও দিচ্ছে। সাকিব ভাবল, দিদিভাইটা বড় খেয়ালি। ওর একটা নিজস্ব
গণ্ডি আছে। ওখানে ও আপনমনে সুখ-দুঃখ সাজায়। সাকিবের তেমন কোনো নিজস্ব
কিছু নেই। সাকিব জলে সাঁতরানোর মতো সবকিছু কাটিয়ে যেতে পারে। কোথাও বাঁধে
না। দিদিভাই পারে না। দিদিভাই ঘাসেও হোঁচট খায়। আর হোঁচট না খেয়ে চলতে
গেলে ওর মনে হয় ও থেমে গেছে। চলতে পারছে না। সোজা পথে চলাটা কি চলা নাকি!

Figure 7.3: Extracted Text

7.3 Experiment using Colored Image taken using Cellphone Camera and Scanned Image from a Scanner

Below is an example of of colored image taken using cellphone camera and scanned image scanned using a scanner from a textbook.

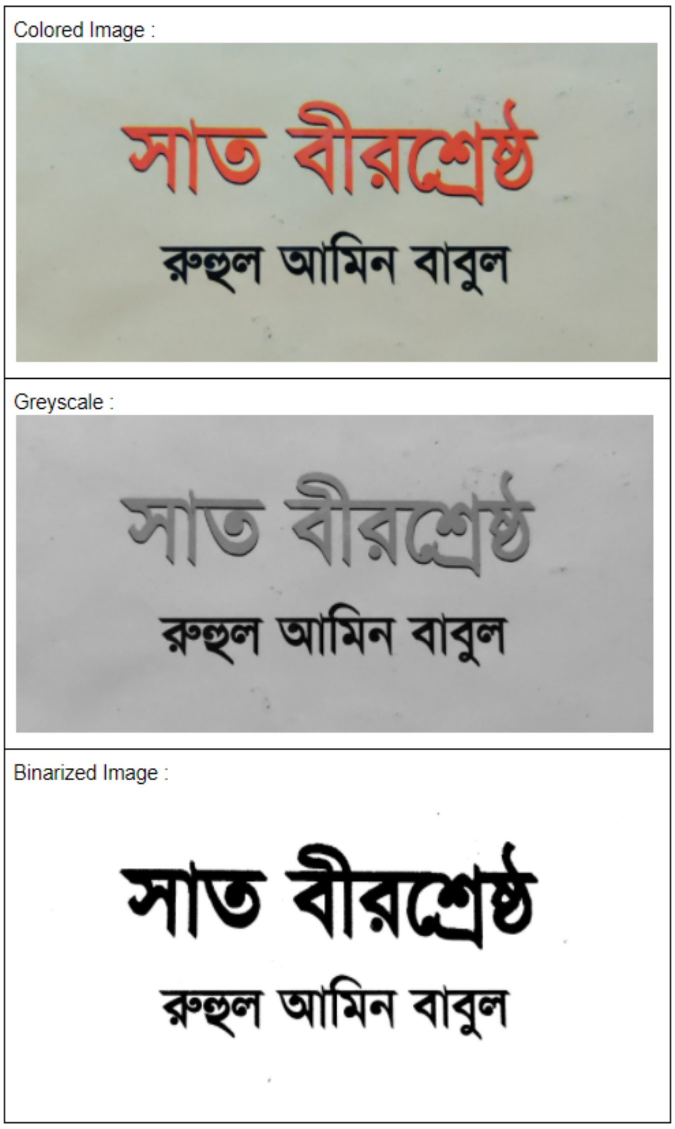
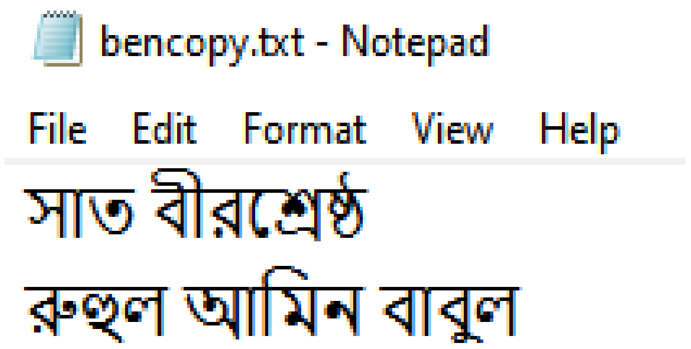
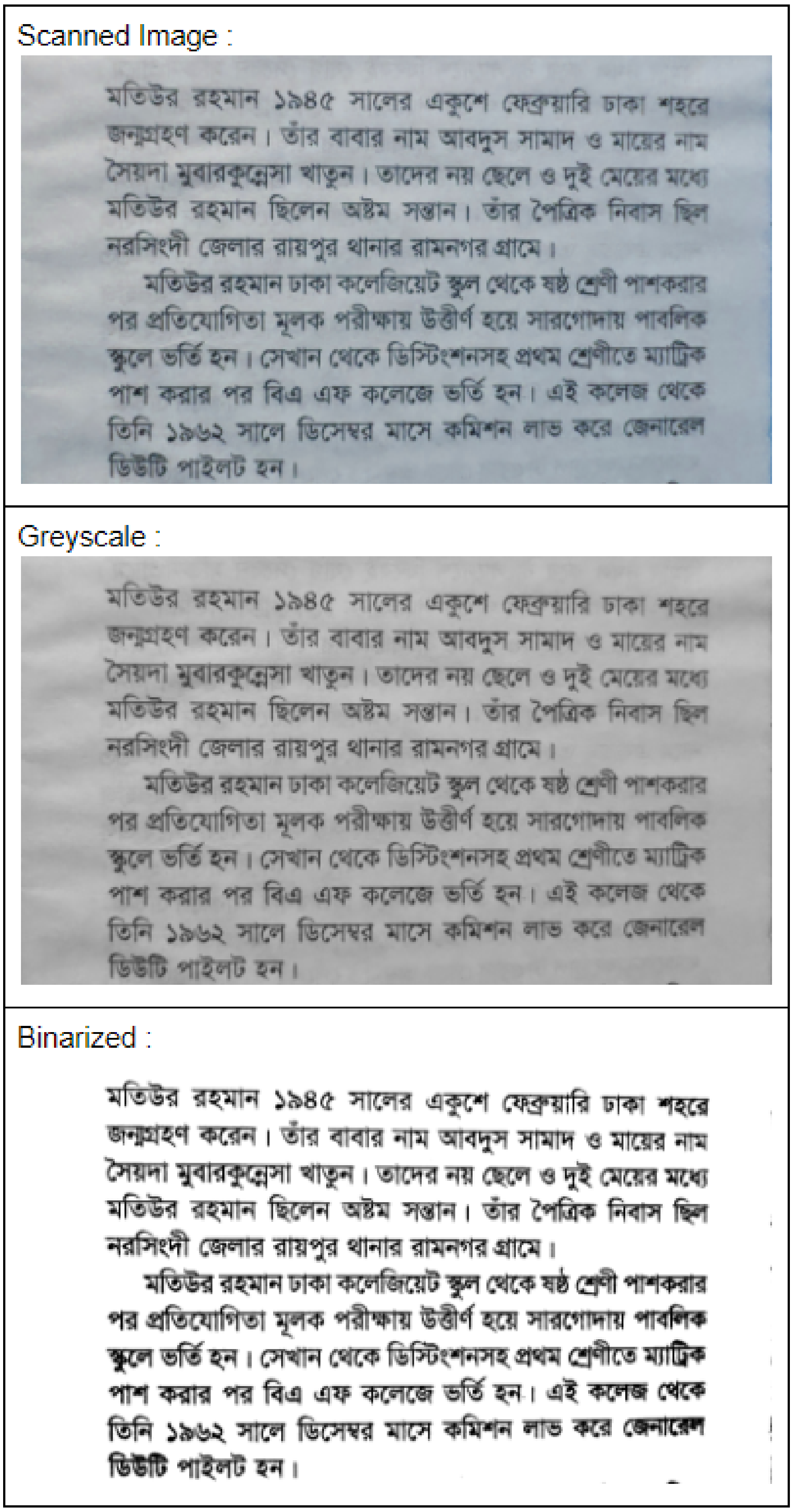


Figure 7.4: Colored image input



```
সাত বীরশ্রেষ্ঠ
রুহুল আমিন বাবুল
```

Figure 7.5: Output from colored image


Scanned Image :

মতিউর রহমান ১৯৪৫ সালের একুশে ফেব্রুয়ারি ঢাকা শহরে
জন্মগ্রহণ করেন। তাঁর বাবার নাম আবদুস সামাদ ও মায়ের নাম
সৈয়দা মুবারকুন্নেসা খাতুন। তাদের নয় ছেলে ও দুই মেয়ের মধ্যে
মতিউর রহমান ছিলেন অষ্টম সন্তান। তাঁর পৈত্রিক নিবাস ছিল
নরসিংদী জেলার রায়পুর থানার রামনগর গ্রামে।
    মতিউর রহমান ঢাকা কলেজিয়েট স্কুল থেকে ষষ্ঠ শ্রেণী পাশকরার
পর প্রতিযোগিতা মূলক পরীক্ষায় উত্তীর্ণ হয়ে সারগোদায় পাবলিক
স্কুলে ভর্তি হন। সেখান থেকে ডিস্টিংশনসহ প্রথম শ্রেণীতে ম্যাট্রিক
পাশ করার পর বিএ এফ কলেজে ভর্তি হন। এই কলেজ থেকে
তিনি ১৯৬২ সালে ডিসেম্বর মাসে কমিশন লাভ করে জেনারেল
ডিউটি পাইলট হন।

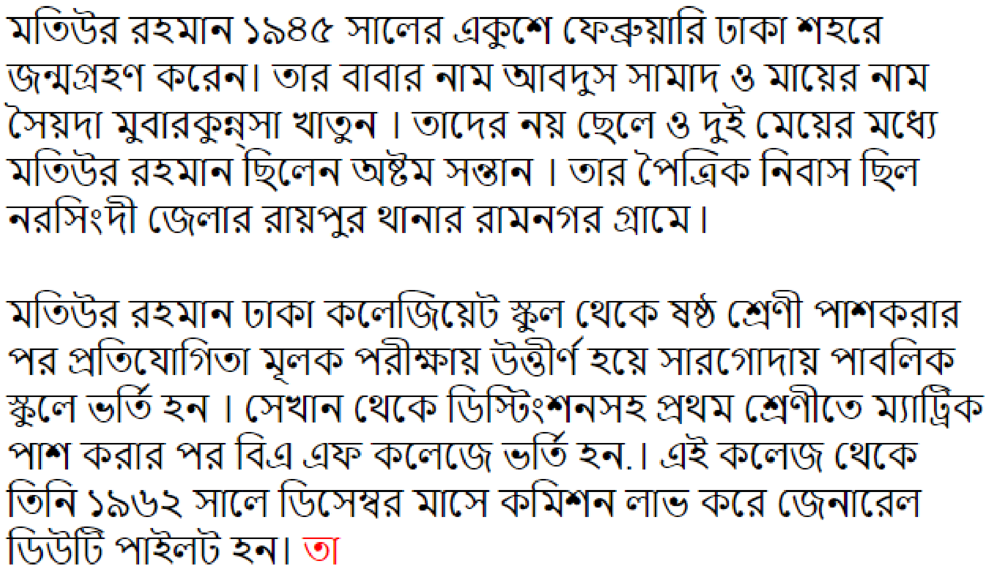
Greyscale :

মতিউর রহমান ১৯৪৫ সালের একুশে ফেব্রুয়ারি ঢাকা শহরে
জন্মগ্রহণ করেন। তাঁর বাবার নাম আবদুস সামাদ ও মায়ের নাম
সৈয়দা মুবারকুন্নেসা খাতুন। তাদের নয় ছেলে ও দুই মেয়ের মধ্যে
মতিউর রহমান ছিলেন অষ্টম সন্তান। তাঁর পৈত্রিক নিবাস ছিল
নরসিংদী জেলার রায়পুর থানার রামনগর গ্রামে।
    মতিউর রহমান ঢাকা কলেজিয়েট স্কুল থেকে ষষ্ঠ শ্রেণী পাশকরার
পর প্রতিযোগিতা মূলক পরীক্ষায় উত্তীর্ণ হয়ে সারগোদায় পাবলিক
স্কুলে ভর্তি হন। সেখান থেকে ডিস্টিংশনসহ প্রথম শ্রেণীতে ম্যাট্রিক
পাশ করার পর বিএ এফ কলেজে ভর্তি হন। এই কলেজ থেকে
তিনি ১৯৬২ সালে ডিসেম্বর মাসে কমিশন লাভ করে জেনারেল
ডিউটি পাইলট হন।

Binarized :

মতিউর রহমান ১৯৪৫ সালের একুশে ফেব্রুয়ারি ঢাকা শহরে
জন্মগ্রহণ করেন। তাঁর বাবার নাম আবদুস সামাদ ও মায়ের নাম
সৈয়দা মুবারকুন্নেসা খাতুন। তাদের নয় ছেলে ও দুই মেয়ের মধ্যে
মতিউর রহমান ছিলেন অষ্টম সন্তান। তাঁর পৈত্রিক নিবাস ছিল
নরসিংদী জেলার রায়পুর থানার রামনগর গ্রামে।
    মতিউর রহমান ঢাকা কলেজিয়েট স্কুল থেকে ষষ্ঠ শ্রেণী পাশকরার
পর প্রতিযোগিতা মূলক পরীক্ষায় উত্তীর্ণ হয়ে সারগোদায় পাবলিক
স্কুলে ভর্তি হন। সেখান থেকে ডিস্টিংশনসহ প্রথম শ্রেণীতে ম্যাট্রিক
পাশ করার পর বিএ এফ কলেজে ভর্তি হন। এই কলেজ থেকে
তিনি ১৯৬২ সালে ডিসেম্বর মাসে কমিশন লাভ করে জেনারেল
ডিউটি পাইলট হন।

Figure 7.6: Input for scanned image



মতিউর রহমান ১৯৪৫ সালের একুশে ফেব্রুয়ারি ঢাকা শহরে
জন্মগ্রহণ করেন। তার বাবার নাম আবদুস সামাদ ও মায়ের নাম
সৈয়দা মুবারকুন্নসা খাতুন । তাদের নয় ছেলে ও দুই মেয়ের মধ্যে
মতিউর রহমান ছিলেন অষ্টম সন্তান । তার পৈত্রিক নিবাস ছিল
নরসিংদী জেলার রায়পুর থানার রামনগর গ্রামে।

মতিউর রহমান ঢাকা কলেজিয়েট স্কুল থেকে ষষ্ঠ শ্রেণী পাশকরার
পর প্রতিযোগিতা মূলক পরীক্ষায় উত্তীর্ণ হয়ে সারগোদায় পাবলিক
স্কুলে ভর্তি হন । সেখান থেকে ডিস্টিংশনসহ প্রথম শ্রেণীতে ম্যাট্রিক
পাশ করার পর বিএ এফ কলেজে ভর্তি হন.। এই কলেজ থেকে
তিনি ১৯৬২ সালে ডিসেম্বর মাসে কমিশন লাভ করে জেনারেল
ডিউটি পাইলট হন। তা

Figure 7.7: Output from the scanned image with error highlighted

Chapter 8

Conclusion and Future Work

8.1 Conclusion

OCR is becoming increasingly popular nowadays. But there are not a lot of Bangla OCR available out there. We wanted to make an OCR that is accurate across several fonts and is easy to use. To do this, we trained our OCR with multiple fonts. The training process is one that is tedious and requires a lot of time and effort. We repeated all of the steps for three different fonts. We experimented with multiple input methods such as Single Line Text image, Multiple Line Text image, Colored Image, Scanned Image. All of them seemed to produce consistent results. During the overall process we faced several challenges starting from Tesseract not working as intended to facing problems with our box files. We proceeded with a positive mindset and with the help of the internet and our beloved teacher, we were able to overcome the challenges. We plan to make our project open source for the public to use and modify. In this way, we intend to contribute to the evergrowning research of Bangla OCR.

8.2 Future Work

Although the entire project was for our thesis project we plan to further work on it and make improvements to the system. Firstly, we would like to include more fonts for training. Fonts which are widely used and popular. We plan to also add support for fonts with varied typefaces and styles over time. Another major improvement would be added dictionary support. A lot of the errors can be fixed by matching words with a dictionary with the highest similarity. Also, with the help of Artificial Intelligence, we can further improve the accuracy of our OCR. Artificial Intelligence makes automatic correction to the text based on the context of a sentence. As Tesseract in itself becomes more and more accurate, we are very hopeful for the future of Bangla OCR.

References

[1] R. Prakashalay, “Pather panchal,” 1929.

[2] Palit, Rajesh, Sattar, and M. Abdus, “Representation of bangla characters in the computer systems,” *1999 Bangladesh Journal of Computer and Information Technology*, 1999.

[3] R. Gonzalez and R. Woods, “Digital image processing,” 2002.

[4] D. Goldman, “Digital image processing,” 2004.

[5] M. A. Hasnat, “Crblp bangla ocr,” 2007. [Online]. Available: https://crblpocr.blogspot.com/.

[6] S. M. M. H. Md. Abul Hasnat and M. Khan, “A high performance domain specific ocr for bangla script,” *2007 Int. Joint Conf. on Computer, Information, and Systems Sciences, and Engineering (CISSE)*, 2007.

[7] R. Smith, “An overview of the tesseract ocr engine,” 2007.

[8] M. A. Hasnat, M. R. Chowdhury, and M. Khan, “An open source tesseract based optical character recognizer for bangla script,” *2009 10th International Conference on Document Analysis and Recognition*, 2009.

[9] ——, “Integrating bangla script recognition support in tesseract ocr,” *Proceedings of the Conference on Language Technology 2009*, 2009.

[10] “Ocropus,” 2009. [Online]. Available: http://code.google.com/p/ocropus/.

[11] S. K. Saha, M. Shamsuzzaman, and M. A. Bhuiyan, “On bangla character recognition,” *2010 13th International Conference on Computer and Information Technology (ICCIT)*, 2010.

[12] F. Y. Omee, S. S. Himel, and M. A. N. Bikas, “A complete workflow for development of bangla ocr,” *International Journal of Computer Applications*, vol. 21, 2011.

[13] A. A. Shinde and D. G. Chougule, “Text pre-processing and text segmentation for ocr,” *2012 International Journal of Computer Science Engineering and Technology (IJCSET)*, 2012.

[14] S. K. A. Hossain and T. Tabassum, “Neural net based complete character recognition scheme for bangla printed text books,” *2014 16th Int’l Conf. Computer and Information Technology*, 2014.

[15] ABBY, “Ocr accuracy measurement,” 2017. [Online]. Available: https://abbyy.technology/en:kb:tip:ocr-accuracy.

[16] B. K. Kuguoglu, “How to use image preprocessing to improve the accuracy of tesseract,” 2018. [Online]. Available: https://www.freecodecamp.org/news/getting-started-with-tesseract-part-ii-f7f9a0899b3f/.

[17] N. Majid and E. H. B. Smith, "Introducing the boise state bangla handwriting dataset and an efficient offline recognizer of isolated bangla characters," *2018 16th International Conference on Frontiers in Handwriting Recognition*, 2018.

[18] Palash, "Training tesseract 4.x (lstm) for bengali," 2020. [Online]. Available: http://palashray.com/training-tesseract-4-x-lstm-for-bengali/.

[19] M. Dausinger, "Improve ocr accuracy with advanced image preprocessing." [Online]. Available: https://docparser.com/blog/improve-ocr-accuracy/#:~:text=Measuring%20OCR%20accuracy%20is%20done,correctly%20(word%20level%20accuracy).

[20] "Open cv image thresholding." [Online]. Available: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[21] "Tesseract user manual." [Online]. Available: https://tesseract-ocr.github.io/tessdoc/.