

Form segmentation, Recover and
Regenerate in English and Bangla
using Tesseract OCR Engine

Tanvir Hasan Reg No: 12101005
Anup Kumar Kar Reg No: 12101033
Sayem Hossain Reg No: 12101046



submitted in partial fulfilment of the degree of
Bachelor of Science, with Honours
at the University of Asia Pacific, Dhaka,
Bangladesh.

April 3, 2016

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Md. Shiplu Hawlader, Lecturer, Department of Computer Science and Engineering, University of Dhaka. We also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

Signature

Md. Shiplu Hawlader
Supervisor

Tanvir Hasan
Candidate

Anup Kumar Kar
Candidate

Sayem Hossain
Candidate

Approval

The Thesis Report "Form segmentation, Recover and Regenerate in English and Bangla using Tesseract OCR Engine" submitted by Tanvir Hasan, Reg.NO.12101005, Anup Kumar Kar, Reg.NO.12101033, Sayem Hossain, Reg.NO.:12101046 students of Spring-2012, to the Department of Computer Science & Engineering, University of Asia Pacific, has been accepted as satisfactory for the partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science & Engineering and approved as to its style and contents.

Approved as to the style & contents by	
Dr. Bilkis Jamal Ferdosi Associate Professor Department of Computer Science & Engineering, University of Asia Pacific	
Dr. Khan Md. Anwarus Salam Assistant Professor Department of Computer Science & Engineering, University of Asia Pacific	
Md. Akhtaruzzaman Adnan Assistant Professor Department of Computer Science & Engineering, University of Asia Pacific	
Md. Habibur Rahman Lecturer Department of Computer Science & Engineering, University of Asia Pacific	

Acknowledgements

First of all, thanks to Almighty Allah for giving us the potency and energy to complete this thesis successfully.

We want to express our gratefulness towards our thesis supervisor Md. Shiplu Hawlader for his valuable advices and important suggestions. His regular and active supervision and erudite directions from the beginning to the end were the driving forces for the successful completion of the research work.

We would like to convey our thankfulness to all of our teachers at the Department of Computer Science and Engineering, University of Asia Pacific. Discussions with many of them have helped us tremendously in improving the quality of our work. We also thank the department for providing us with resources which were necessary for the preparation of the thesis.

And last but not the least, we would like to express thanks to our parents and family members for their tremendous support and inspiration.

Abstract

Image segmentation is an important component in many image analysis and computer vision tasks. Particularly, the problem of efficient interactive foreground object segmentation in still images is of great practical importance in image editing and has been the interest of research for a long time. Classical image segmentation tools use either texture, color or edge (contrast) information for the purpose of segmentation. Deformable models, Graph-cut, GrabCut etc. are some prominent methods used for the segmentation of a foreground object. Object segmentation methods have helped in many computer vision areas, such as scene representation & interpretation, content based image retrieval, object tracking in videos, medical applications etc.

Most object segmentation techniques in computer vision are based on the principle of boundary detection. These segmentation techniques assume a significant and constant gray level change between the object(s) of interest and the background. However, this is not true in the case of textured images. In textured images, there exist many local edges of the texture micro units (texels), due to the basic nature of a texture image. In case of textured images, the object boundary is defined as the place where texture property changes. So to perform the correct segmentation in case of textured images, there is a need to incorporate the textural information in the segmentation process.

Contents

1	Introduction	1
1.1	Objective	2
1.2	Motivation	3
1.3	Overview of this book	3
2	Background Study	5
2.1	Gray Scale	5
2.2	Blur	5
2.2.1	Gaussian smoothing	6
2.3	Thresholding	6
2.4	Edge Detection	7
2.5	Contour	8
2.6	Tesseract OCR engine	9
2.7	Why Tesseract	10
2.8	Optical character recognition systems	10
3	Literature Review	11
3.1	Canny Edge Detection	11
3.1.1	Canny Edge Detection Algorithm	11
3.2	Contours	14
3.3	Detecting shapes in images	15
3.3.1	Detection quadrilaterals	15
3.3.2	Detection Rectangles, squares, equilateral	16
3.4	Tesseract OCR Engine	17
3.4.1	What is Tesseract?	17
3.4.2	Architecture	17
3.4.3	Working of Tesseract	18
3.4.4	How does Tesseract work?	19
3.4.5	Limitations of Tesseract	20
3.5	Training Tesseract	20
3.5.1	Generating training images	20
3.5.2	Make box files	20
3.5.3	Run Tesseract	21
3.5.4	Compute character set	21
3.6	OpenCV	21

4 Proposed Method	22
4.1 Data Extraction	22
4.1.1 Convert RGB to Gray Image	22
4.1.2 Image Smoothing	23
4.1.3 Thresholding	23
4.1.4 Edge Detection	24
4.1.5 Finding Contour	24
4.1.6 Select only rectangle shape region	24
4.1.7 Store only unique region	25
4.1.8 Mapping From Field and Value	25
4.2 Data Reorganization	25
4.3 Tesseract Training	26
5 Performance Analysis	27
5.1 Environments of Experiments	27
5.2 Performance for English form	28
5.2.1 Sample form 1	28
5.2.2 Sample form 2	31
5.2.3 Sample form 3	34
5.2.4 Sample form 4	37
5.2.5 Sample form 5	40
5.3 Performance for Bangla form	42
5.3.1 Sample form 1	43
5.3.2 Sample form 2	45
5.3.3 Sample form 3	47
5.3.4 Sample form 4	49
5.3.5 Sample form 5	51
6 Conclusion	54
6.1 Discussion	54
6.2 Future Works	55
Bibliography	56

List of Tables

3.1	Edge Ditection	13
5.1	Comparison between Input and Output frequency of Sample Input 1	29
5.2	Comparison between Input and Output frequency of Sample Input 2	32
5.3	Comparison between Input and Output frequency of Sample Input 3	35
5.4	Comparison between Input and Output frequency of Sample Input 4	38
5.5	Comparison between Input and Output frequency of Sample Input 5	41

List of Figures

2.1	Gaussian smoothing	6
2.2	Thresholding	7
2.3	Edge Detection	8
2.4	Find Contour	9
3.1	Discrete Approximation to Gaussian function with $\sigma = 1.4$	12
3.2	Find Courours	15
3.3	Detection of quadrilaterals	16
3.4	Tesseract Flow	18
4.1	Form template	22
4.2	RGB to Gray	23
4.3	Image after applying gaussian blur in sample form	23
4.4	Thresholded image after appling thresholding in sample form	24
4.5	Canny Edge detected image after applying canny edge detection in sam- ple form	24
5.1	Sample English form 1	28
5.2	Bar chart Input Output Frequency of Sample form 1	30
5.3	Sample English form 2	31
5.4	Bar chart Input Output Frequency of Sample form 2	33
5.5	Sample English form 3	34
5.6	Bar chart Input Output Frequency of Sample form 3	36
5.7	Sample English form 4	37
5.8	Bar chart Input Output Frequency of Sample form 4	39
5.9	Sample English form 5	40
5.10	Bar chart Input Output Frequency of Sample form 4	42
5.11	Sample Bangla form 1	43
5.12	Comparison between Input and Output frequency of Sample Input 1 . .	44
5.13	Bar chart Input Output Frequency of Sample form 1	45
5.14	Sample Bangla form 2	45
5.15	Comparison between Input and Output frequency of Sample Input 2 . .	46
5.16	Bar chart Input Output Frequency of Sample form 2	47
5.17	Sample Bangla form 3	47
5.18	Comparison between Input and Output frequency of Sample Input 3 . .	48
5.19	Bar chart Input Output Frequency of Sample form 3	49
5.20	Sample Bangla form 4	49

5.21 Comparison between Input and Output frequency of Sample Input 4	50
5.22 Bar chart Input Output Frequency of Sample form 4	51
5.23 Sample Bangla form 5	51
5.24 Comparison between Input and Output frequency of Sample Input 5	52
5.25 Bar chart Input Output Frequency of Sample form 5	53

Chapter 1

Introduction

Segmentation of images [1] is the widely investigated field of image processing, image analysis and important module of early vision problem. It is the process to cluster a form image into some isolated image regions corresponding to individual surface, objects or some natural part of the object. It is the process of separating an image into some disjoint or distinct regions whose characteristic such as intensity; color texture etc. are similar. No two such regions are similar with respect to these characteristics. In digital image processing, digital image analysis usually involves a "low-level" and a "high-level" processing. In low-level analysis, the representation of an image is transformed from a numerical array of pixel intensities to a symbolic set of image primitives: edges and regions. In high-level analysis, object labels (or interpretations) are assigned to these primitives, thereby providing a semantic description of the image.

An image is segmented [1] for different kind of implementations like object recognition to extract data from it, occlusion boundary estimation within motion or stereo systems, image compression, image editing or image database lookup.

The output of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region are similar with respect to some characteristic or computed an image is segmented for different kind of implementations like object recognition to extract data from it, occlusion boundary estimation within motion or stereo systems, image compression, image editing or image database lookup. Adjacent regions are significantly different with respect to the same characteristic.

Image segmentation is a fundamental part of the "low level" aspects of computer visions and has many practical applications such as in medical imaging, industrial automation and satellite imagery. Traditional methods for image segmentation have ap-

proached the problem either from localization in class space using region information or from localization in position, using edge or boundary information. for monochrome images generally are based on one of two basic properties of gray- level values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in gray level. The principal areas of interest within this category are detection of isolated points and detection of lines and edges in an image. The principal approaches in the first category are based on edge detection, and boundary detection. Basically, the idea underlying most edge-detection techniques is the computation of a local derivative operator. The first derivative of the gray-level profile is positive at the leading edge of a transition, negative at the trailing edge, and zero in areas of constant gray level. Hence the magnitude of the first derivative can be used to detect the presence of an edge in an image. In this paper we are proposing a form segmentation, an image segmentation methodology using threshold techniques that will be applied to a form image and data matching algorithms to detect different objects and clusters. The goal of segmenting a form image is to extract text data isolating texts of different part of that image by segmenting form into several sections. It becomes easy to extract digital data by processing individual objects into texts if we can first isolate that object scenario and use it for processing.

1.1 Objective

The objective of this thesis is to decompose the image into parts for further analysis. In simple cases, the environment might be well enough controlled so that the segmentation process reliably extracts only the parts that need to be analyzed further. The segmentation is reliable provided that the person's clothing or room background does not have the same color components as a human face. In complex cases, such as extracting a complete road network from a grayscale aerial image, the segmentation problem can be very difficult and might require application of a great deal of domain building knowledge. Another objective of segmentation is to perform a change of representation. The pixels of the image must be organized into higher-level units that are either more meaningful or more efficient for further analysis (or both). A critical issue is whether or not segmentation can be performed for many different domains using general bottom-up methods that do not use any special domain knowledge. This chapter presents segmentation methods that have potential use in many different domains. Both region-based and curve-based units are discussed in the following sections. The

prospects of having a single segmentation system work well for all problems appear to be dim. Experience has shown that an implementor of machine vision applications must be able to choose from a toolset of methods and perhaps tailor a solution using knowledge of the application.

1.2 Motivation

In recent years, with the advancement of digital era, we are facing a problem of converting handwritten data into digital data for the purpose of saving them in database or repositories in order to analyse or perform operation on the data or keeping log for the future. We still need to process handwritten documents and forms manually that takes lot of times and resources and increases the possibilities to make mistake when processing that data. That causes serious inefficiency and pain worth not to do it manually.

Several offices including government and non-government organizations gets a lot of applications and different kind of forms every day that needs to be processed immediately with efficient measure and more accuracy. In this paper we have proposed an efficient way to process those analogue data into digital texts through form segmentation, a technique of processing form image into digital text. Experimental results shows it's accuracy and efficiency to process a form that is lot less time consuming and effective.

1.3 Overview of this book

In this chapter we have introduced the image segmentation, form segmentation, discuss our objective and motivation. Rest of the chapters are organized as follows.

In Chapter 2, we will acquire some background knowledge by discussing about image processing technique, algorithm and tesseract OCR Engine.

In Chapter 3, we are going to discuss about some existing algorithm and technique of Image Processing, tesseract ocr engine working procedure and image processing tool openCV.

In chapter 4, we proposed a working method. In this chapter we discuss about Data Extraction, Data Reorganization and Tesseract Training.

In chapter, 5, we discuss about the performance of our method and output of some sample form.

Finally chapter 6, concludes the document.

Chapter 2

Background Study

Base of a good research is the understanding of the background terms and definition. To understand the image processing and character reorganisation, one have to clearly understand about image filtering, image transformations and tesseract OCR Engine operation. In this chapter as background knowledge discovery we will discuss about some image processing term and tesseract.

2.1 Gray Scale

In photography and computing, a grayscale or greyscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest. [2]

2.2 Blur

Blurring is a very powerful operation used in image processing and procedural texture generation. Blurs involve calculating weighted averages of areas of pixels in a source image for each pixel of the final blurred image. Blurring images so they become fuzzy may not seem like a useful operation, but actually is very useful for generating background effects and shadows[3]. In our form image blur operation increase border edges.

2.2.1 Gaussian smoothing

The Gaussian smoothing operator is a 2-D convolution operator that is used to ‘blur’ images and remove detail and noise. The idea of Gaussian smoothing is to use this 2-D distribution as a ‘point-spread’ function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. This Gaussian smoothing technique is widely used effect in graphics software, typically to reduce image noise and reduce detail. [3]

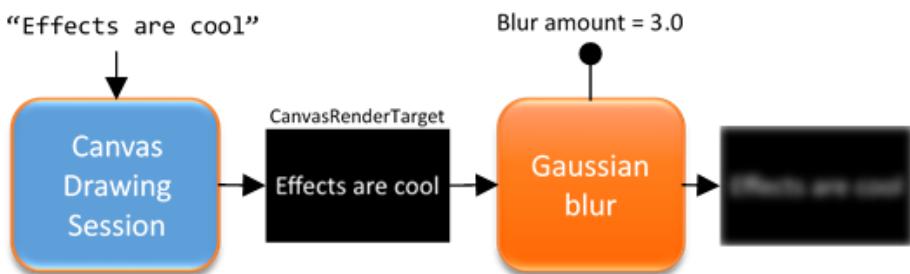


Figure 2.1: Gaussian smoothing

2.3 Thresholding

Thresholding is one of the most basic techniques for what is called Image Segmentation. When the threshold technique is applied on an image, we get segments inside the image and each of segments are represent something.[4]

- Thresholding is a simple way of segmentation.
- We separate out various regions of an image regarding to objects which we want to analyze. The object pixels and the background pixels are the backbone of this separation.
- To differentiate the pixels we are interested in from the rest, we perform a comparison of each pixel intensity value with respect to a threshold.
- Once we have separated properly the important pixels, we can set them with a determined value to identify.

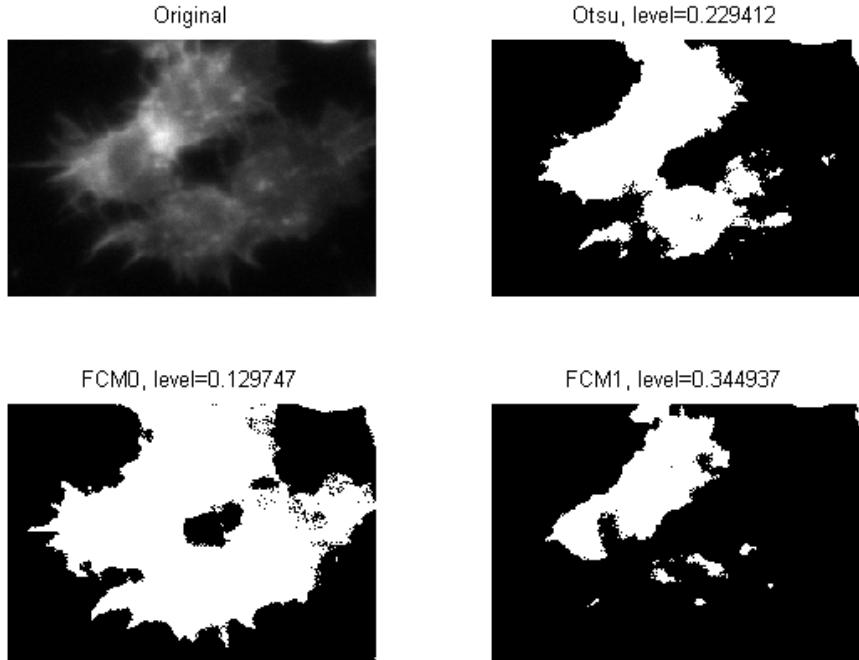


Figure 2.2: Thresholding

2.4 Edge Detection

In digital image processing, edge detection is an important subject matter. Edge detection is a crucial step in object recognition. It is a process of finding sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. In short, the goal of edge detection is to produce a line drawing of the input image. [5]

The Canny operator is also known as the optimal detector, developed by John F. Canny in 1986. There are multiple steps to implement the Canny operator. First, a Gaussian filters is used to smooth the image to remove noise in an image. Second, compute the gradient magnitude. Third, apply the algorithm to remove the pixels that are not part of an edge. Last step is involved with the use of hysteresis thresholding along edges. Hysteresis is based on two thresholds which are upper and lower. If a pixel gradient is higher than the upper threshold, then the pixel will be marked as an edge and if a pixel gradient is below the lower threshold, then the pixel will be marked as a non-edge[5] [6].

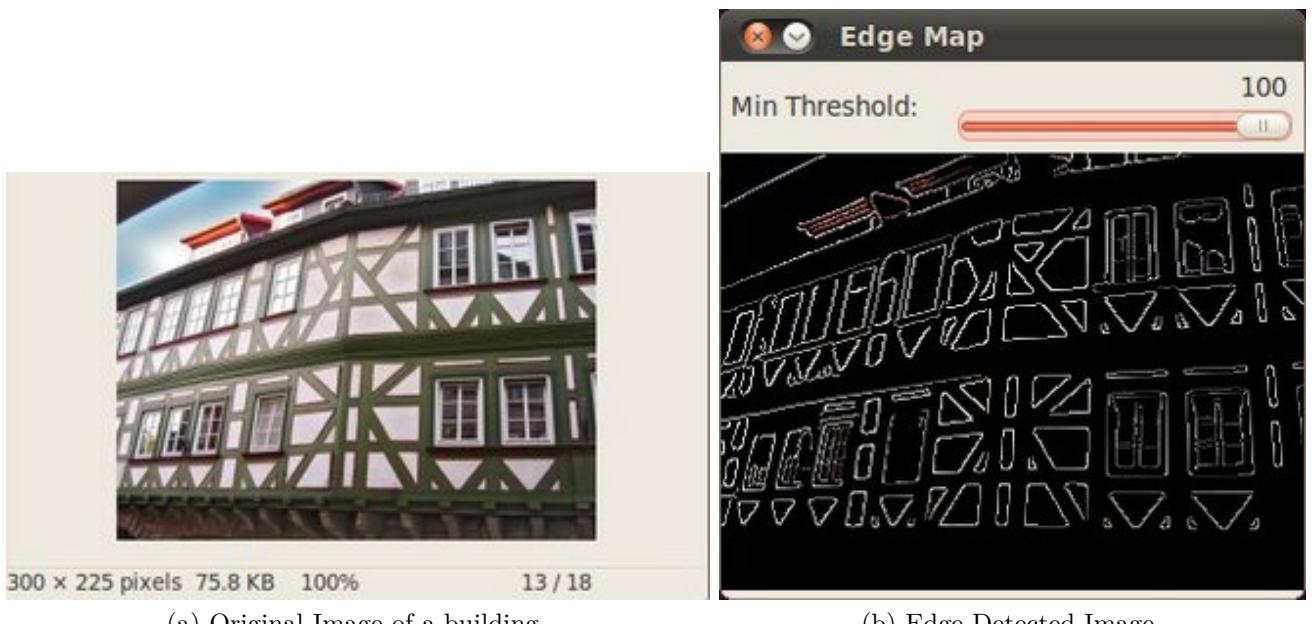


Figure 2.3: Edge Detection

2.5 Contour

Although algorithms like the Canny edge detector can be used to find the edge pixels that separate different segments. The next step is to be able to assemble those edge pixels into contours. A contour is a list of points that represent, in one way or another, a curve in an image. This representation can be different depending on the circumstance at hand. There are many ways to represent a curve. Contours are represented in OpenCV by sequences in which every entry in the sequence encodes information about the location of the next point on the curve.

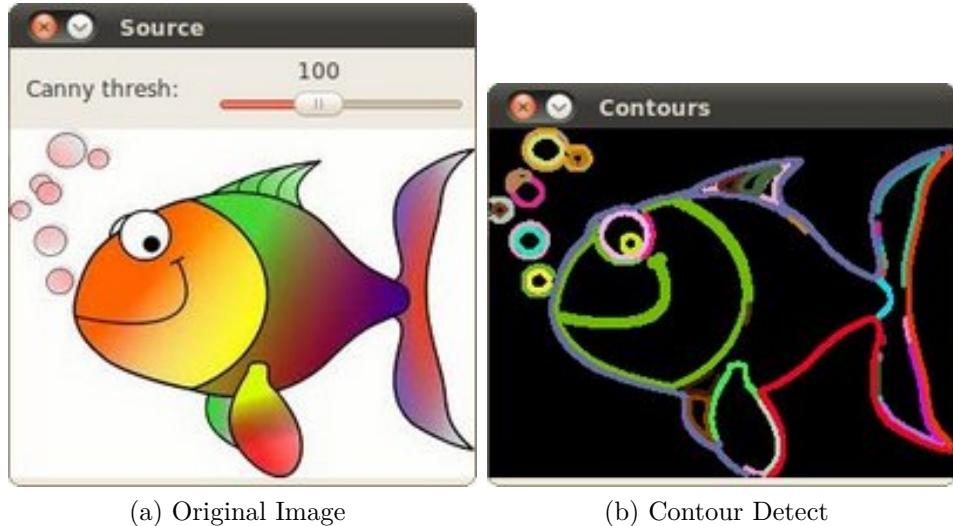


Figure 2.4: Find Contour

2.6 Tesseract OCR engine

Tesseract is an open source optical character recognition (OCR) engine. HP was originally started it as a project. Later it was modified, improved and taken over by Google and later released as open source in year 2005[7] [8].

Tesseract is considered as one of the most accurate free software OCR engines currently available. A large variety of other OCR software now uses it as a base. It is an excellent quality OCR program, with a large amount of flexibility, a solid codebase, and a large, engaged community of interested people around it.

This thesis investigates the principles of optical character recognition used in the Tesseract OCR engine and techniques to improve its efficiency and runtime.

Optical character recognition (OCR) method has been used in converting printed text into editable text in various applications over a variety of devices such as Scanners, computers, tablets etc. But now Mobile is taking over the computer in all the domains but OCR still remains one not so conquered field. This paper focuses on improving the Tesseract OCR efficiency for Bangla.

This thesis presents a preprocessing technique being applied to the Tesseract Engine to improve the recognition of the characters keeping the runtime low.

2.7 Why Tesseract

The first relevant criterion in Tesseract is the fact that it is free and open source (FOSS), which is an advantage and a key point in the research development.

Usually, whenever Tesseract is compared to another free OCR tool, it is the best whether in terms of recognition rate or speed. Even, when it is compared with the Finereader commercial tool, Tesseract arrives to rub it and managed to overtake for handwritten writing.

2.8 Optical character recognition systems

An OCR system is a system that takes a text image as input and applies certain treatments through modules making up the system in order to output editable file with the same text.

The architecture of an OCR system varies from one system to another as needed. Optical Character Recognition systems are usually composed of the following phases:

- Preprocessing phase: prepares the sensor data to the next phase. It is a set of treatments allowing image quality increasing.
- Segmentation phase: delimits document elements (line, word, character). By applying good segmentation techniques, we can increase the performance of OCR systems.
- Feature extraction phase: defines features characterizing the delimited elements of a document. Feature extraction is one of the most important steps in developing a classification system. This step describes the various features characterizing the delimited elements of a document.
- Classification phase: recognizes and identifies each element. It is performed based on the extracted features.
- Post-processing phase: it is an optional phase. It may be automatic or manual.

Several approaches and techniques have been developed for each module.

Chapter 3

Literature Review

The From segmentation process is related to edge detection, finding contours and contours shapes, and text detection related to Tesseract and Tesseract training. Thus in this chapter we will discuss about some state of the art image processing techniques and Tesseract OCR applications.

3.1 Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works [5] [9].

3.1.1 Canny Edge Detection Algorithm

Canny edge detection algorithm go through these following steps:

1. In order to implement the canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges. And because the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of the Gaussian mask, the lower is the detector's sensitivity to noise. The localization error in the detected

edges also increases slightly as the Gaussian width is increased. The Gaussian mask used in my implementation is shown below.

	2	4	5	4	2
	4	9	12	9	4
$\frac{1}{115}$	5	12	15	12	5
	4	9	12	9	4
	2	4	5	4	2

Figure 3.1: Discrete Approximation to Gaussian function with $\sigma = 1.4$

- After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). They are shown below:

-1	0	+1
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

The magnitude, or EDGE STRENGTH, of the gradient is then approximated using the formula: $|G| = |G_x| + |G_y|$

- Finding the edge direction is trivial once the gradient in the x and y directions are known. However, you will generate an error whenever sumX is equal to zero. So in the code there has to be a restriction set whenever this takes place. Whenever the gradient in the x direction is equal to zero, the edge direction has to be equal

to 90 degrees or 0 degrees, depending on what the value of the gradient in the y-direction is equal to. If GY has a value of zero, the edge direction will equal 0 degrees. Otherwise the edge direction will equal 90 degrees. The formula for finding the edge direction is just:

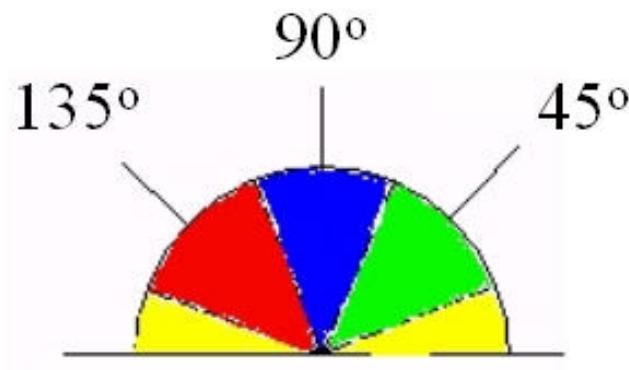
$$\theta = \tan^{-1}(Gy/Gx)$$

4. Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a 5x5 image are aligned as follows:

x	x	x	x	x
x	x	x	x	x
x	x	x	x	x
x	x	a	x	x
x	x	x	x	x
x	x	x	x	x

Table 3.1: Edge Detection

Then, it can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels - 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal). So now the edge orientation has to be resolved into one of these four directions depending on which direction it is closest to (e.g. if the orientation angle is found to be 3 degrees, make it zero degrees). Think of this as taking a semicircle and dividing it into 5 regions.



Therefore, any edge direction falling within the **yellow range** (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the **green range** (22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the **blue range** (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the **red range** (112.5 to 157.5 degrees) is set to 135 degrees.

5. After the edge directions are known, nonmaximum suppression now has to be applied. Nonmaximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. This will give a thin line in the output image.
6. Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T_1 is applied to an image, and an edge has an average strength equal to T_1 , then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses 2 thresholds, a high and a low. Any pixel in the image that has a value greater than T_1 is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T_2 are also selected as edge pixels. If you think of following an edge, you need a gradient of T_2 to start but you don't stop till you hit a gradient below T_1 . [10][11]

3.2 Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. [12]

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- `findContours` function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be

black.

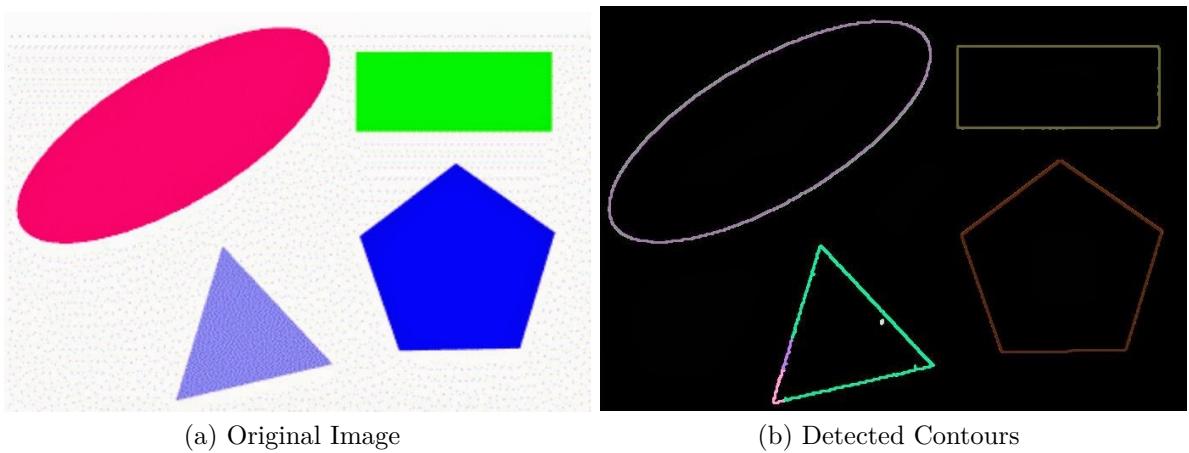


Figure 3.2: Find Courours

3.3 Detecting shapes in images

Doing image processing and especially blob analysis it is often required to check some objects' shape and depending on it perform further processing of a particular object or not. For example, some applications may require finding only rectangle from all the detected objects, or quadrilaterals, rectangles, etc.

3.3.1 Detection quadrilaterals

Detection of quadrilaterals and triangles has pretty much the same idea - we are checking mean distance between provided shape's edge pixels and the edge of estimated quadrilateral/triangle. The only difference here is the way how to estimate parameters of the shape we want to recognize and how to check distance to the estimated shape.

Let's start with quadrilaterals first. For a given shape we can make an assumption that it is a quadrilateral, find its four corners and then using similar method as we've used for circles we can check if our assumption is correct or not - check how good the shape fits into the quadrilateral with assumed parameters.



Figure 3.3: Detection of quadrilaterals

As we can see on the image above, we may have different objects and quadrilateral finder provides different results for them. For shapes which really look like quadrilateral, the quadrilateral finder is able to find their corners more or less correctly (problem may occur in the case if object has rounded corners). But for other types of objects (circles, stars, etc.), quadrilateral finder does not provide any good result. And this is correct, since this routine supposes the given shape is really quadrilateral.

Now, when we made the assumption that a subjected object is quadrilateral and we got its corner, we just need to see how good is the fit of the object into the quadrilateral with those found corner - we need to make sure there are no edge points of the given shape which are too far away from the edge of the assumed quadrilateral.

3.3.2 Detection Rectangles, squares, equilateral

Once we made a check for quadrilateral shape and got its 4 corners, we may do some further checks to detect sub-type of the quadrilateral: trapezoid, parallelogram, rectangle, rhombus or square. These checks are based on checking angles between opposite/adjacent sides and their length. First we check two opposite sides - if they are parallel, then we get at least trapezoid. Then we check two other sides - if they are parallel two, then we have parallelogram. If two adjacent sides are perpendicular, then we got rectangle. The last check is to compare length of two adjacent sides - if they are equal, then parallelogram becomes rhombus and rectangle becomes square. And of course we need to add small possible error, so if angle between lines equals to 2 degrees, they are still treated as parallel. [13]

3.4 Tesseract OCR Engine

3.4.1 What is Tesseract?

An Overview of the Tesseract OCR Engine describes Tesseract as: "Tesseract is an open source optical character recognition(OCR) engine. HP originally was originally started it as a project. Later it was modified, improved and taken over by Google and later released as open source in year 2005. It is now available at" (Smith, 2007) . It is very portable as compared to others and supports various platforms. Its focus is more towards providing less rejection and improved accuracy. Currently only command base version is available but there are many projects with UI built on top of it which could be forked. As of now Tesseract version 3.02 is released and available for use. Now Tesseract is developed and maintained by Google. It provides support for around 139 languages [7].

3.4.2 Architecture

Tesseract OCR is an elegant engine with various layers. It works in step by step manner as shown in the block diagram in fig 3.4. The first step in the cycle is to sense the color intensities of the image, named as adaptive thresholding, and converts the image into binary images. Second step is to do the connected component analysis of the image, which does the task of extracting character outlines. This step is the main process of this cycle as it does the OCR of image with white text and blacks rest of the image. Tesseract was probably the first to use these cycles to process the input image. After this the outlines extracted from image are converted into Blobs(Binary Long Objects). It is then organized as lines and regions and further analysis is for some fixed area. After extraction the extracted components are chopped into words and delimited with spaces. Recognition in text then starts which is a two pass process. As shown in fig 1, the first part is when attempt to recognize each word is made. Each satisfactory word is accepted and second pass is started to gather remaining words. This brings in the role of adaptive classifier. The adaptive classifier then will classify text in more accurate manner. The adaptive classifier needs to be trained beforehand to work accurately. When the classifier receives some data, it has to resolve the issues and assign the proper place of the text[7] [14].

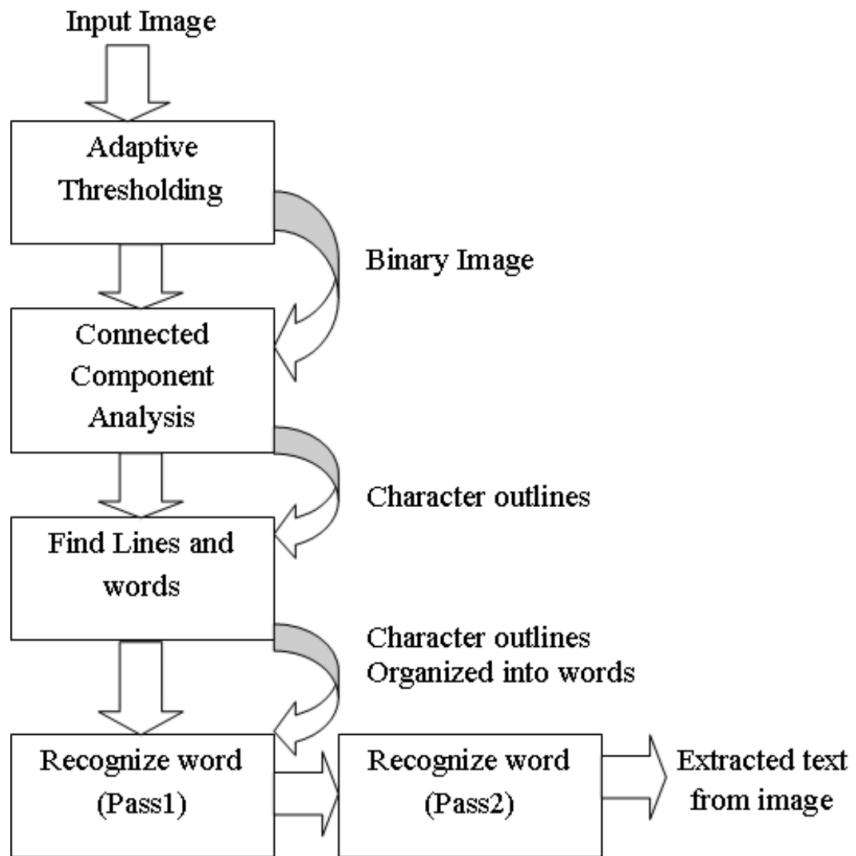


Figure 3.4: Tesseract Flow

3.4.3 Working of Tesseract

Tesseract works pretty much as a scanner. Its interface is pretty simple as it takes input o the command lines with very basic commands. We need to input any image with text in it.

The basic Tesseract command takes only two arguments: First is input image that contains text and second argument is output text file which is usually text file.

Tesseract by default picks the extension of output file as .txt. There no need to specify the explicitly the output file extension.Tesseract supports various languages. Each language comes with a trained language data file.

The language file must be kept in a location Tesseract knows. When using in the project it is advised to keep it within the project folder. This folder is Tesseract home folder in your machine. In this research, we are aiming to extract English and Bangla characters from the images so we have to keep both Bangla and English data files [7].

3.4.4 How does Tesseract work?

The following is a brief overview of how Tesseract works[8]:

1. Outlines are analysed and stored
2. Outlines are gathered together as Blobs
3. Blobs are organized into text lines
4. Text lines are broken into words
5. First pass of recognition process attempts to recognize each word in turn
6. Satisfactory words passed to adaptive trainer
7. Lessons learned by adaptive trainer employed in a second pass, which attempts to recognize the words that were not recognized satisfactorily in the first pass
8. Fuzzy spaces resolved and text checked for small caps
9. Digital texts are outputted

During these processes, Tesseract uses:

1. algorithms for detecting text lines from a skewed page
2. algorithms for detecting proportional and non proportional words (a proportional word is a word where all the letters are the same width)
3. algorithms for chopping joined characters and for associating broken characters
4. linguistic analysis to identify the most likely word formed by a cluster of characters
5. two character classifiers: a static classifier, and an adaptive classifier which employs training data, and which is better at distinguishing between upper and lower case letters

3.4.5 Limitations of Tesseract

Tesseract is an OCR engine, not a complete OCR program.[8] Tesseract is an OCR engine rather than a fully featured program similar to commercial OCR software such as Nuance's Omnipage. It was originally intended to serve as a component part of other programs or systems. Although Tesseract works from the command line, to be usable by the average user the engine must be integrated into other programs or interfaces, such as FreeOCR.net, WeOCR or OCRpous. Without integration into programs such as these, Tesseract has no page layout analysis, no output formatting and no graphical user interface (GUI).

3.5 Training Tesseract

The setup for English & Bangla OCR within Tesseract is another important task. The way train the engine and input the dataset matters a lot[15][16]. Training Tesseract for English & Bangla includes four steps mainly and they are as follows:

3.5.1 Generating training images

Tesseract official website explains these steps very well "The first step is to determine the full character set to be used, and prepare a text or word processor file containing a set of examples". The two important points to remember for a training file are: Firstly make sure the file contains all the characters that we are expecting. Secondly there should be at least 5 samples. There should be more samples of the more frequent characters - at least 20. Keeping all this in mind let's explore how do we create box files [15].

3.5.2 Make box files

Box file is defined as a sample to match with the characters. We need a 'box' file for each training image. Tesseract official website defines the box file as "The box file is a text file that lists the characters in the training image, in order, one per line, with the coordinates of the bounding box around the image". Tesseract is not so intuitive about the sample data. Inconsistencies may lead to wrong interpretation of data[15].

3.5.3 Run Tesseract

In this step we run the Tesseract engines with the new training files. The purpose is to create the trained dataset which works as a rule engine. Also it creates log files[15].

3.5.4 Compute character set

Guidelines from Tesseract website state that "Tesseract needs to know the set of possible characters it can output. To generate the unichar set data file, use the unicharset_extractor program on the box files generated above":extractor language. fontname. box One more requirement in this is, Tesseract needs to have access to character properties i.e. isalpha, isdigit, isupper, islower, ispunctuation. The system described in have classified Hindi OCR and its setup, training very nicely. The only limitation is that the system is for desktop. We have to build the similar system for mobile devices.[15]

3.6 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions and algorithms. Their main focus is to provide API mainly aimed at real-time computer vision. It was originally developed by Intel. OpenCV library is free for use for development (under the open source BSD license). The best feature is that the library is cross-platform . [15]

Chapter 4

Proposed Method

In this thesis we have implemented a well define digital process to recognize hand writings from a registration form or any kind of forms which are mostly documentation of government or private institutions. Image recognition is a process, which usually consist of taking a picture, process the image then present the results. The whole work can be divided into two parts:

4.1 Data Extraction

We work on a defined format of from. In this process the from every field should be separate by box region.

Name :	Tanvir Hasan	Roll :	5
Reg :	12101005	Sec :	A
Programme :	Bsc. Engineering	Dept. :	Computer Science and Engineering
Semester :	First	Year :	First
Total Credit :	21	Total Credit complete :	21

Figure 4.1: Form template

4.1.1 Convert RGB to Gray Image

First we have to take input form as picture. Then convert RGB image to gray image.

Name :	Tanvir Hasan	Roll :	5
Reg :	12101005	Sec :	A
Programme :	Bsc. Engineering	Dept. :	Computer Science and Engineering
Semester :	First	Year :	First
Total Credit :	21	Total Credit complete :	21

Figure 4.2: RGB to Gray

4.1.2 Image Smoothing

Smoothing, also called blurring, is a simple and frequently used image processing operation. There are many reasons for smoothing, but it is usually done to reduce noise or camera artefacts. Smoothing is also important when we wish to reduce the resolution of an image in a principled way[17]. For our thesis work we used Gaussian blur and this is probably the most useful though not the fastest.

Name :	Tanvir Hasan	Roll :	5
Reg :	12101005	Sec :	A
Programme :	Bsc. Engineering	Dept. :	Computer Science and Engineering
Semester :	First	Year :	First
Total Credit :	21	Total Credit complete :	21

Figure 4.3: Image after applying gaussian blur in sample form

4.1.3 Thresholding

Next we work on image thresholds. Frequently we have done many layers of processing steps and want either to make a final decision about the pixels in an image or to categorically reject those pixels below or above some value while keeping the others. The OpenCV function `Threshold()` accomplishes these tasks. The basic idea is that an array is given, along with a threshold, and then set a value for every element of the array depending on whether it is below or above the threshold[17].

Name :	Tanvir Hasan	Roll :	5
Reg :	12101005	Sec :	A
Programme :	Bsc. Engineering	Dept. :	Computer Science and Engineering
Semester :	First	Year :	First
Total Credit :	21	Total Credit complete :	21

Figure 4.4: Thresholded image after applying thresholding in sample form

4.1.4 Edge Detection

We work on the process of edge recognition of an image. For this we used canny edge detector. The method just described in chapter 3.1 for finding edges. The Canny() function expects an input image, which must be gray scale, and an output image, which must also be gray scale (but which will actually be a Boolean image).[17]

Name :	Tanvir Hasan	Roll :	5
Reg :	12101005	Sec :	A
Programme :	Bsc. Engineering	Dept. :	Computer Science and Engineering
Semester :	First	Year :	First
Total Credit :	21	Total Credit complete :	21

Figure 4.5: Canny Edge detected image after applying canny edge detection in sample form

4.1.5 Finding Contour

After edge detection we find out the contours on these defined edges. For this we using opencv function findContours().

4.1.6 Select only rectangle shape region

There can be many regular and irregular shape of contours. But we need only rectangular shape contour. After find contours we filter out irregular shape and keep only the rectangular shape contours. In the 3.3 section we describe how to find rectangle shape.

4.1.7 Store only unique region

For thicker edge one region can occur multiple times. For this reason we need to filter out only unique region. To do this we map these regions with their left most point coordinate x,y. In every selected left most x,y coordinate we must find unique rectangle region. This is the most efficient way. Otherwise mapping whole region point will be costly.

4.1.8 Mapping From Field and Value

For mapping from different type of field with value, we first store every unique region left-upper most x,y 2D point. Then we sort these 2D points respectively y coordinate then x coordinate. Then we take pair of regions from sorted regions. The first element of pair will be from field and second field will be value.

4.2 Data Reorganization

After extracting data from an analogue image, we work on recognizing our extracted data by using Tesseract open source engine. Tesseract converts the input image into binary format using thresholding. Outlines of components are stored for connected component analysis. Nesting of outlines is done which gathers the outlines together to form a Blob. Text lines are analyzed for fixed pitch and proportional text. Then the lines are broken into words by analysis according to the character spacing. Fixed pitch is chopped in character cells and proportional text is broken into words by definite spaces and fuzzy spaces[18].

Tesseract recognizes a word in two passes, one tries to recognize the words in the first pass. If the match is found, then the found word is passed on to the adaptive classifier, which recognizes the text more accurately. During the second pass, the words which were not at all recognized or were not well recognized in the first pass are recognized again through a run over through the page. Finally Tesseract resolves fuzzy spaces. To locate small and capital text, Tesseract checks alternative hypothesis for x-height OCR engine[18] [8].

4.3 Tesseract Training

For better accuracy we need to train tesseract OCR engine with different type of data. The training procedure of tesseract is well describe in official website of tesseract[19][18]. In short, the training step go through these following steps:

1. Generate Training Images
2. Make Box Files
3. Edit Box Files
4. Run Tesseract for Training
5. Compute the Character Set
6. set unicharset properties
7. font properties
8. Clustering
9. Putting it all together and make tessdata file.

Chapter 5

Performance Analysis

As we have described the proposed method in the chapter 4. In this chapter we will discuss about result and performance.

5.1 Environments of Experiments

Text examples in public datasets usually occur within high-quality (high-resolution, well-focused) imagery. In our setting, text often occurs at lower-resolution and with significant blur. Our focus is to achieve text spotting in a real-time system moving through an environment. We first examine how much the information is attached with an analogue image than we detect the text and extract the data. Next, we train the data. Finally, we evaluate the accuracy. We tested with many different categories of the images such as images are taken with mobile camera, scanning images, DSLR camera and handy cam. The accuracy of the output varied greatly. Accuracy varies mainly for image quality. An image is taken with DSLR camera produces most accurate results for it's picture quality. Image of DSLR camera is less noise free. So, it's easy to extract data and OCR can detect the data perfectly. For handy cam images accuracy is slightly get down. Because picture quality is not good as DSLR camera and there is more noise. If we use scanned images, it it produces more accurate result than handy cam. Because there is less noise than handy cam images. But for mobile camera images the accuracy level is not so good as regarding the other three image categories. For mobile camera there is more noise and image quality is not good.

5.2 Performance for English form

We test our proposed algorithm in different types of sample form. The result depends on the tesseract training data. For english we train "Arial", "Courier", "Calibri", "Times New Roman" fonts. Here we show our some output result and analysis of our training data.

5.2.1 Sample form 1

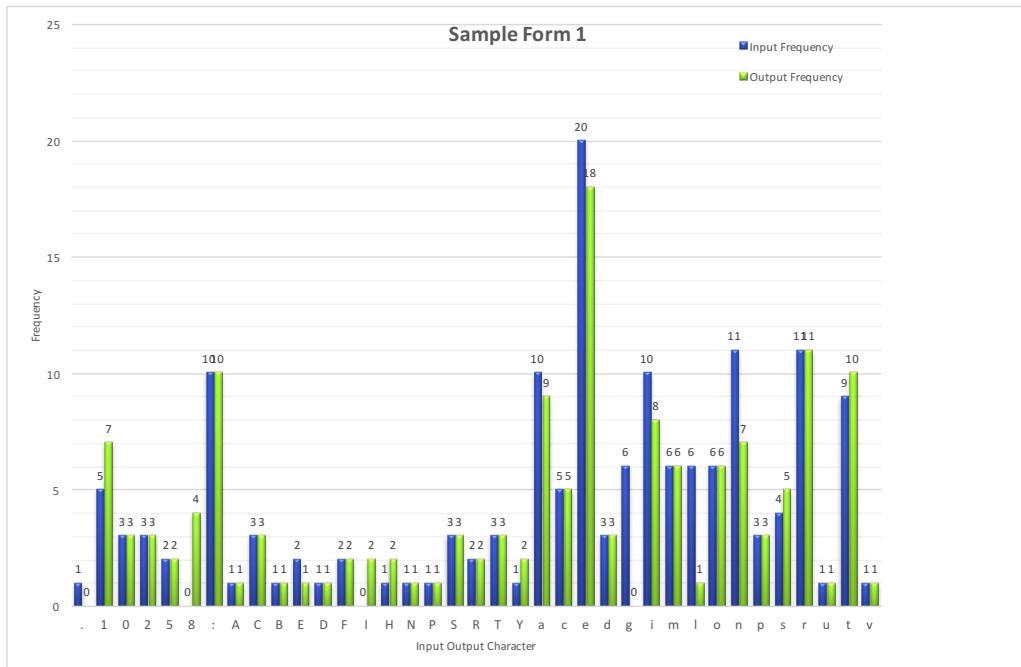


Figure 5.1: Sample English form 1

character	Input Frequency	Output Frequency
0	3	3
1	5	7
2	3	3
5	2	2
8	0	4
:	10	10
A	1	1
B	1	1
C	3	3
D	1	1
E	2	1
F	2	2
G	1	2
H	0	2
I	1	1
N	1	1
P	2	2
R	3	3
S	3	3
T	2	2
Y	9	9
a	5	5
c	3	3
d	18	18
e	10	8
i	2	1
l	6	6
m	8	7
o	6	6
p	3	3
r	11	11
s	5	5
t	10	10
u	1	1
v	1	1

Table 5.1: Comparison between Input and Output frequency of Sample Input 1

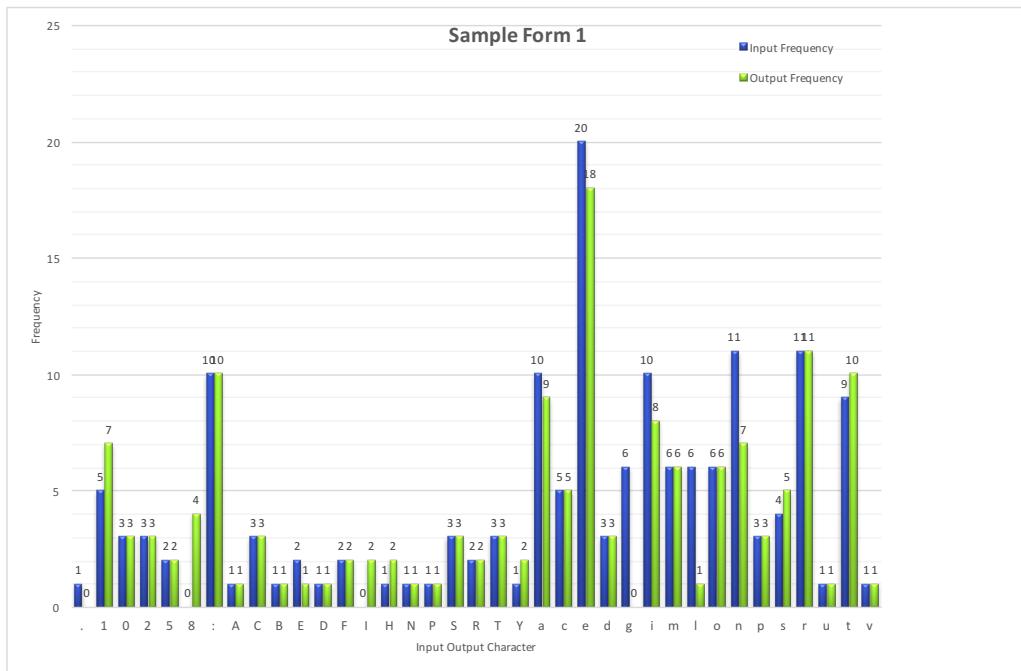


Figure 5.2: Bar chart Input Output Frequency of Sample form 1

5.2.2 Sample form 2

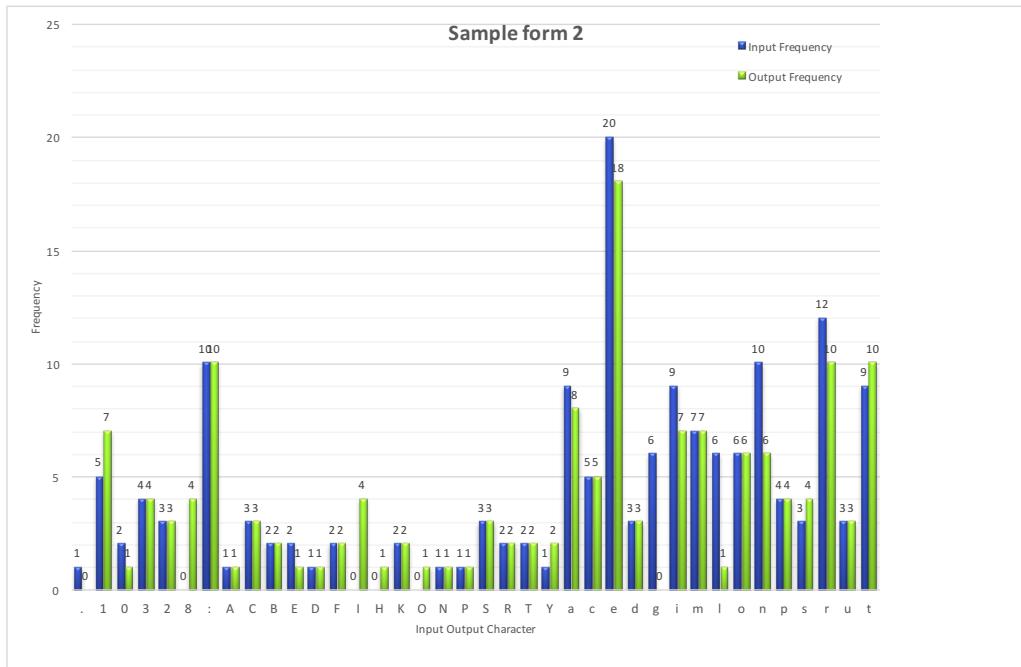


Figure 5.3: Sample English form 2

character	Input Frequency	Output Frequency
0	2	1
1	5	7
2	3	3
3	4	4
8	0	4
:	10	10
A	1	1
B	2	2
C	3	3
D	1	1
E	2	1
F	2	2
H	0	1
I	0	4
K	2	2
N	1	1
O	0	1
P	1	1
R	2	2
S	1	3
T	2	2
Y	1	2
a	7	8
c	4	5
d	3	3
e	18	18
i	10	7
l	5	1
m	7	7
n	7	6
o	5	6
p	12	4
r	1	10
s	3	4
t	9	10
u	3	3

Table 5.2: Comparison between Input and Output frequency of Sample Input 2

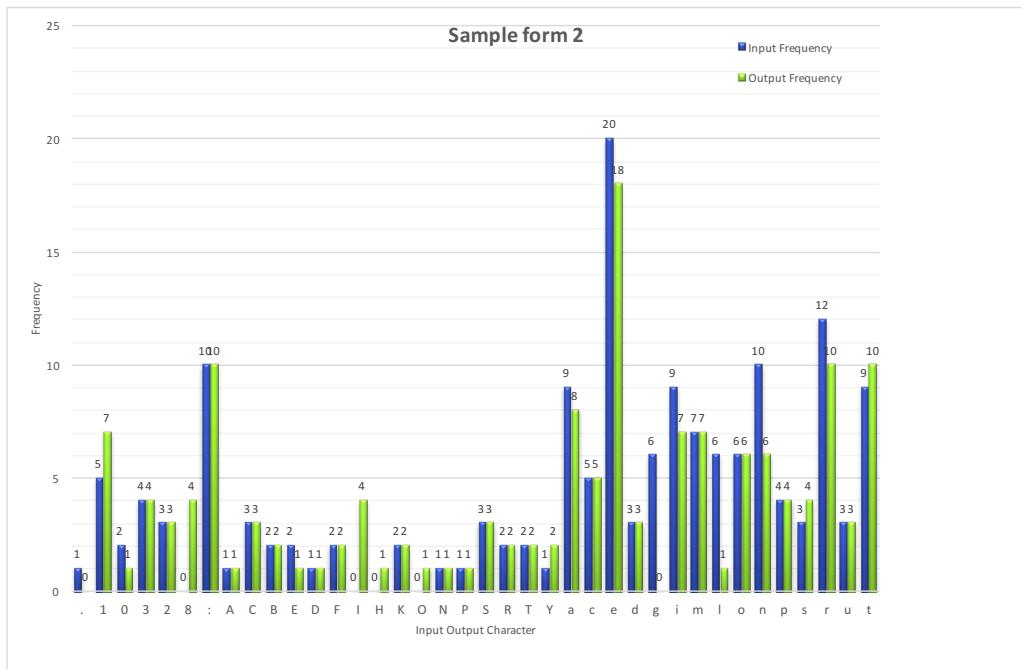


Figure 5.4: Bar chart Input Output Frequency of Sample form 2

5.2.3 Sample form 3

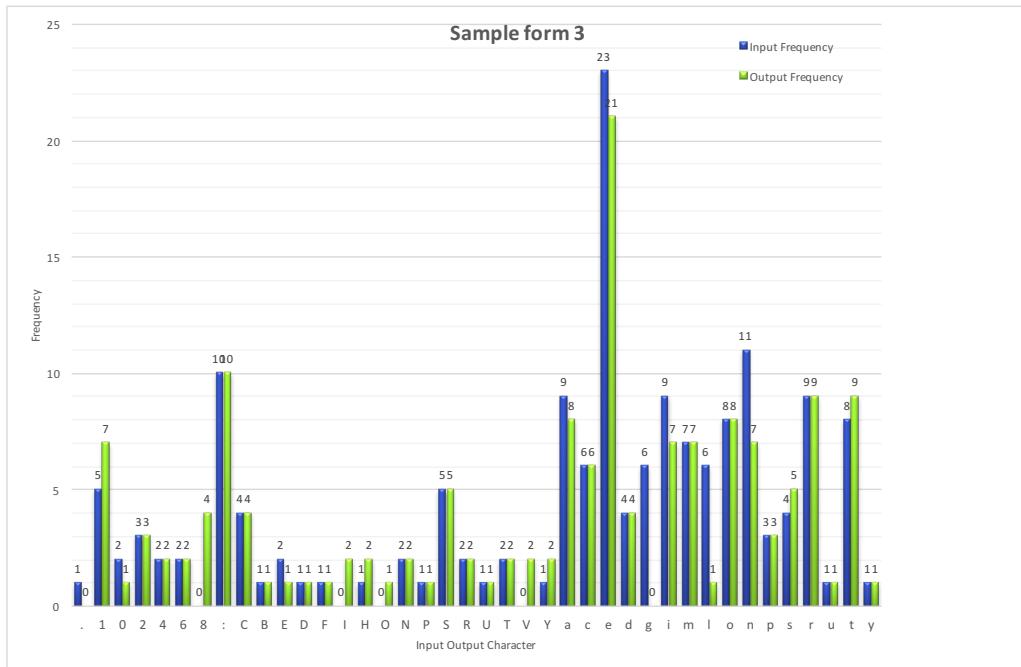


Figure 5.5: Sample English form 3

character	Input Frequency	Output Frequency
0	2	1
1	5	7
2	3	3
4	2	2
6	2	2
8	0	4
:	10	10
B	1	1
C	4	4
D	1	1
E	2	1
F	1	1
H	1	2
I	0	2
N	2	2
O	0	1
P	1	1
R	2	2
S	5	5
T	2	2
U	1	1
V	0	2
Y	1	2
a	7	8
c	6	6
d	4	4
e	21	21
i	9	7
l	4	1
m	7	7
n	7	7
o	8	8
p	3	3
r	9	9
s	5	5
t	9	9
u	1	1
y	1	1

Table 5.3: Comparison between Input and Output frequency of Sample Input 3

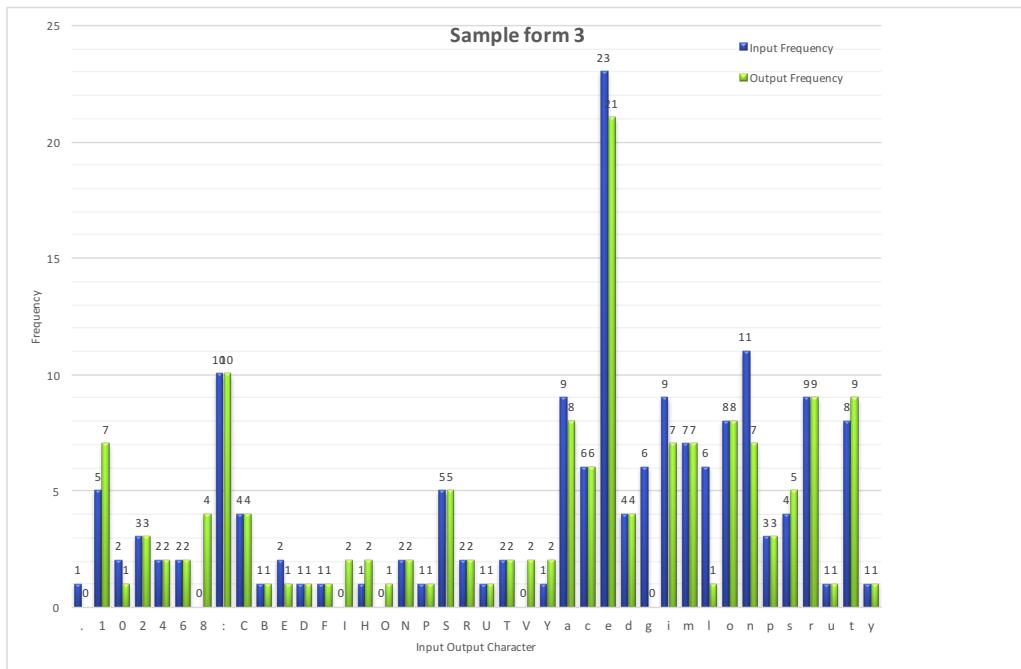


Figure 5.6: Bar chart Input Output Frequency of Sample form 3

5.2.4 Sample form 4

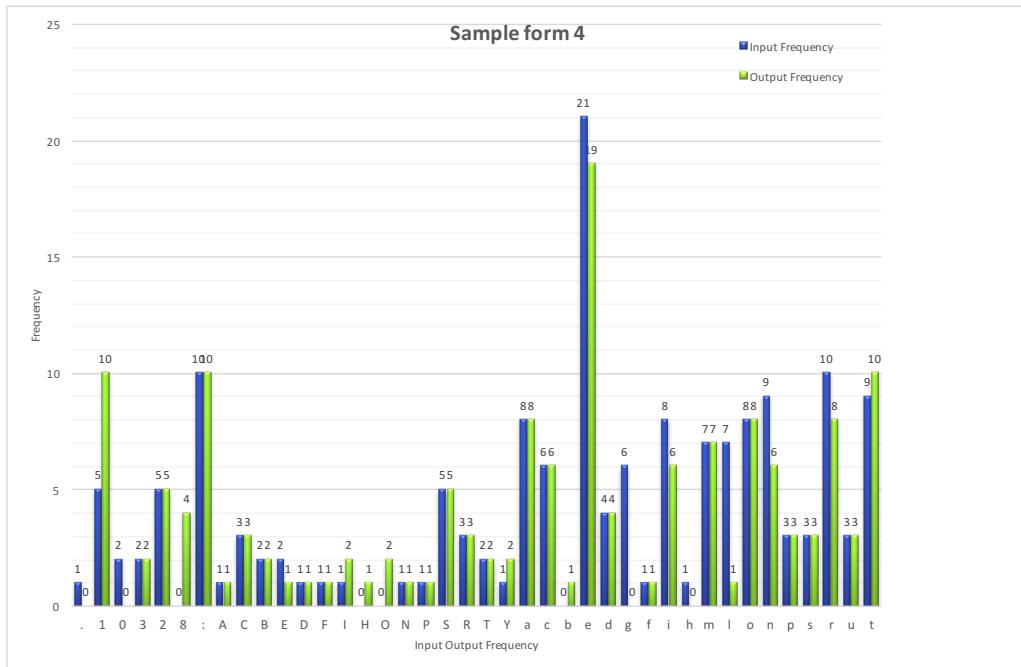


Figure 5.7: Sample English form 4

character	Input Frequency	Output Frequency
1	4	10
2	5	5
3	2	2
8	0	4
:	10	10
A	1	1
B	1	2
C	3	3
D	1	1
E	1	1
F	1	1
H	1	1
I	2	2
N	1	1
O	2	2
P	1	1
R	3	3
S	5	5
T	2	2
Y	2	2
a	8	8
b	1	1
c	c	6
d	4	4
e	17	19
f	1	1
i	6	6
l	1	1
m	7	7
n	6	6
o	2	8
p	3	3
r	8	8
s	3	3
t	9	10
u	3	3

Table 5.4: Comparison between Input and Output frequency of Sample Input 4

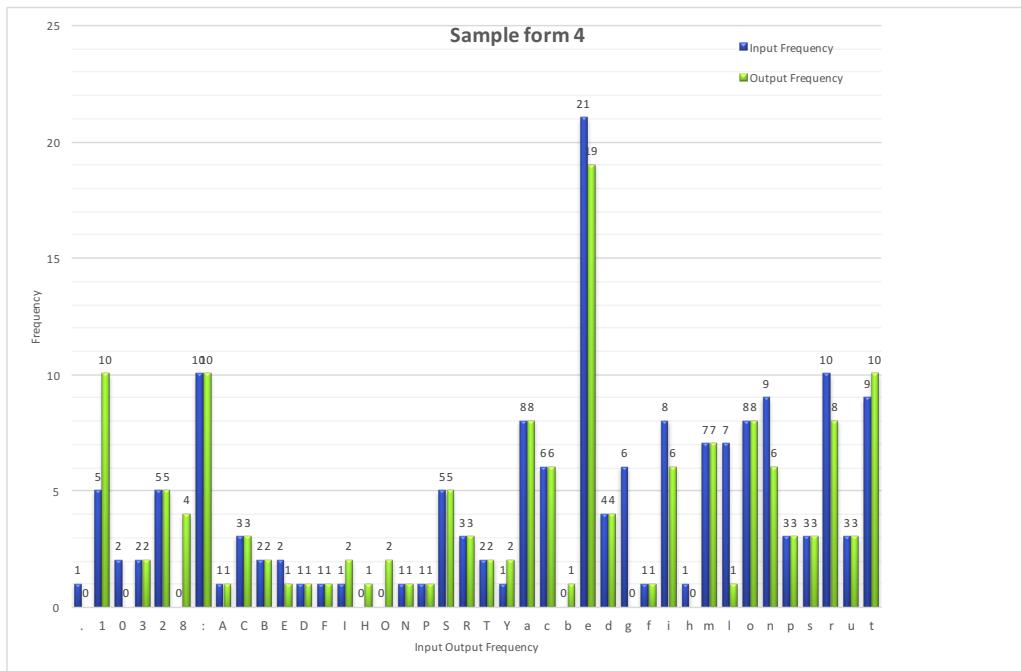


Figure 5.8: Bar chart Input Output Frequency of Sample form 4

5.2.5 Sample form 5

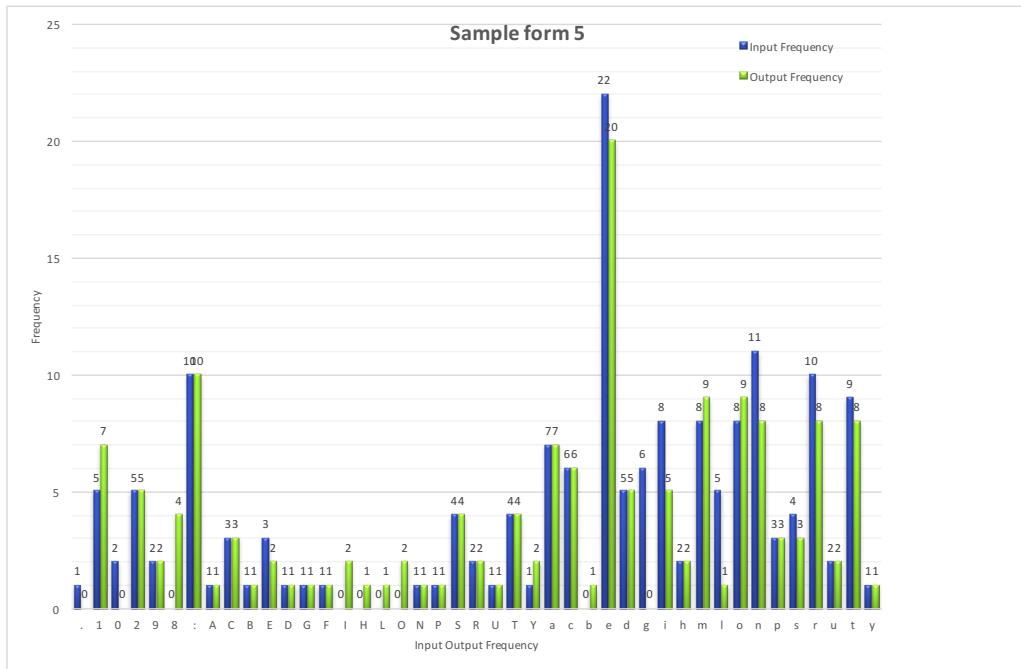


Figure 5.9: Sample English form 5

character	Input Frequency	Output Frequency
.	1	0
1	5	7
0	2	0
2	5	5
9	2	2
8	0	4
:	10	10
A	1	1
C	3	3
B	1	1
E	3	2
D	1	1
G	1	1
F	1	1
I	0	2
H	0	1
L	0	1
O	0	2
N	1	1
P	1	1
S	4	4
R	2	2
U	1	1
T	4	4
Y	1	2
a	7	7
c	6	6
b	0	1
e	22	20
d	5	5
g	6	0
i	8	5
h	2	2
m	8	9
l	5	1
o	8	9
n	11	8
p	3	3
s	4	3
r	10	8
u	2	2
t	9	8
y	1	1

Table 5.5: Comparison between Input and Output frequency of Sample Input 5

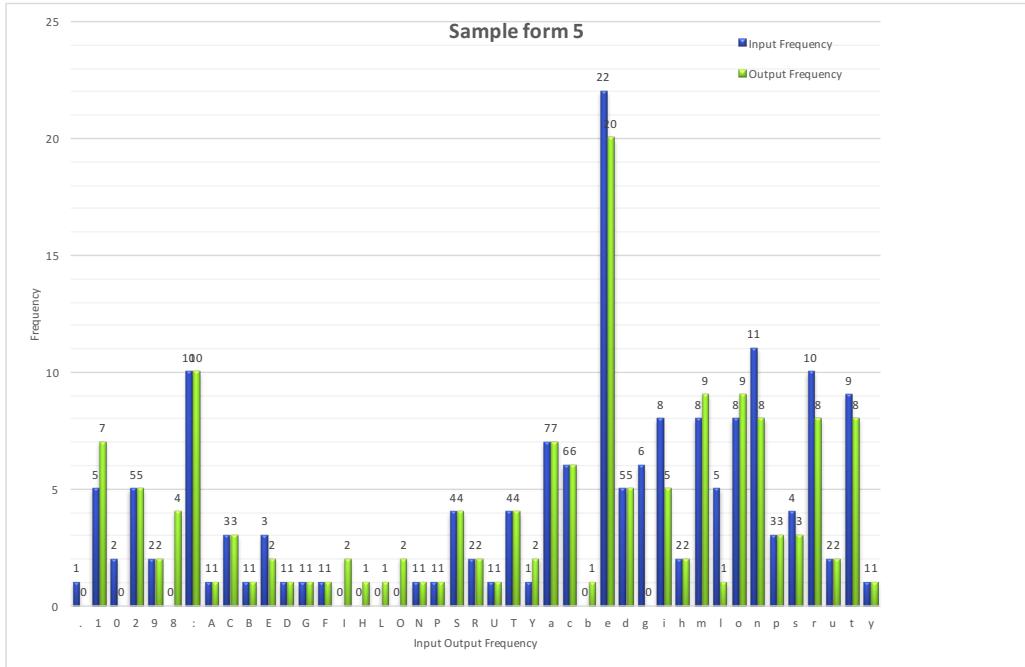


Figure 5.10: Bar chart Input Output Frequency of Sample form 4

According to these frequency table and bar chart we can say our trained tesseract accuracy is upto 85%. Here accuracy of character 'g' is below average. Most of the time character 'g' recognize as digit '8'. And double 'l' character (Ex: Roll) recognize as 'H'. It can be resolve with more frequent fonts training.

5.3 Performance for Bangla form

The result of Bangla form depends on the tesseract training data same as English form. For Bangla we train "Siyam rupali" font. Here we show our some output result and analysis of our training data.

5.3.1 Sample form 1

নামঃ	তানভির হাসান অনিক
রোলঃ	১২১০১০০৫
বিভাগঃ	সিএসই
বিশ্ববিদ্যালয়ঃ	ইউএপি
শিক্ষাবর্ষঃ	২০১৬

Figure 5.11: Sample Bangla form 1

Character	Input Frequency	Output Frequency
ং	3	5
অ	1	1
ই	1	2
ও	1	1
এ	2	2
ক	2	2
গ	1	1
চ	1	0
ত	1	1
'	8	0
ৰ	2	3
ন	4	4
প	1	1
ভ	2	2
ব	6	5
য	2	1
ম	1	1
০	4	0
ল	4	2
ষ	2	2
শ	2	2
হ	1	1
স	1	3
;	1	0
ঁ	1	0
ি	8	7
া	7	7
ি	0	1
ঁ	1	1
়	6	4
য	0	1
ঁ	4	4
ঁ	0	4
ঁ	2	2
ঁ	1	1
ঁ	1	1
ঁ	1	1

Figure 5.12: Comparison between Input and Output frequency of Sample Input 1

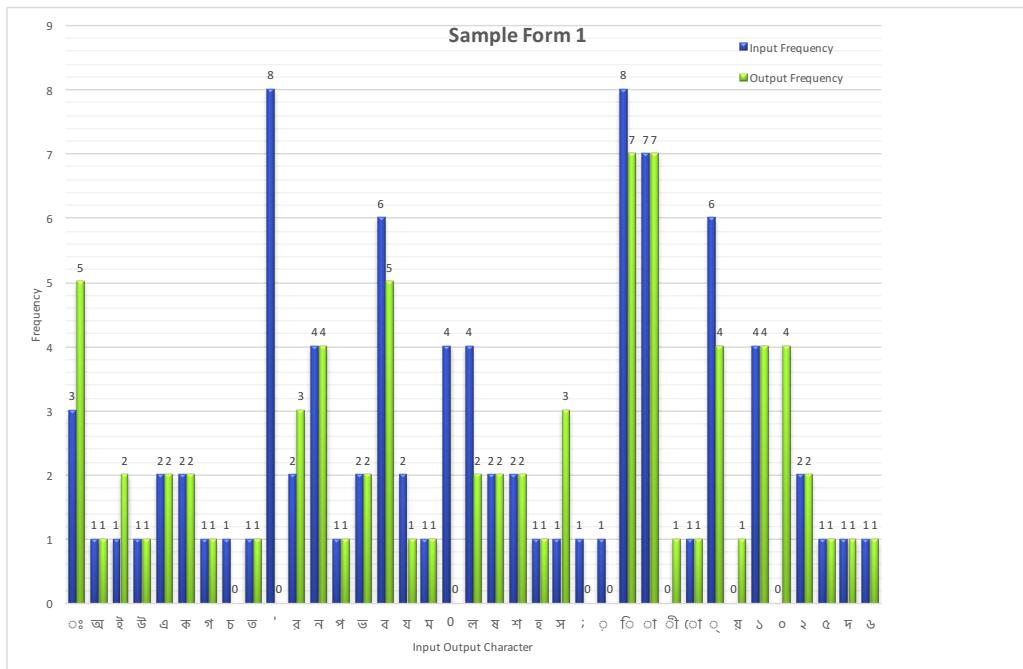


Figure 5.13: Bar chart Input Output Frequency of Sample form 1

5.3.2 Sample form 2

নামঃ	সাইফুল ইসলাম
রোলঃ	১২১০১০৩২
বিভাগঃ	সিএসই
বিশ্ববিদ্যালয়ঃ	ইউএপি
শিক্ষাবর্ষঃ	২০১৬

Figure 5.14: Sample Bangla form 2

Character	Input Frequency	Output Frequency
ং	4	5
ৰ	4	4
৳	0	1
়	1	0
এ	2	2
ক	1	1
গ	1	1
,	1	0
চ	1	0
'	7	0
দ	1	1
ন	1	1
ফ	1	1
প	1	1
ভ	1	1
ব	6	5
য	2	1
ম	2	2
র	1	2
ল	5	4
ষ	2	2
শ	2	2
স	4	4
ঁ	1	0
ି	6	6
ଠ	6	6
ୟ	1	1
ୱ	1	0
୳	1	1
୥	6	4
୯	0	1
୧	4	4
୦	3	3
୭	1	1
୨	3	3
୫	1	1

Figure 5.15: Comparison between Input and Output frequency of Sample Input 2

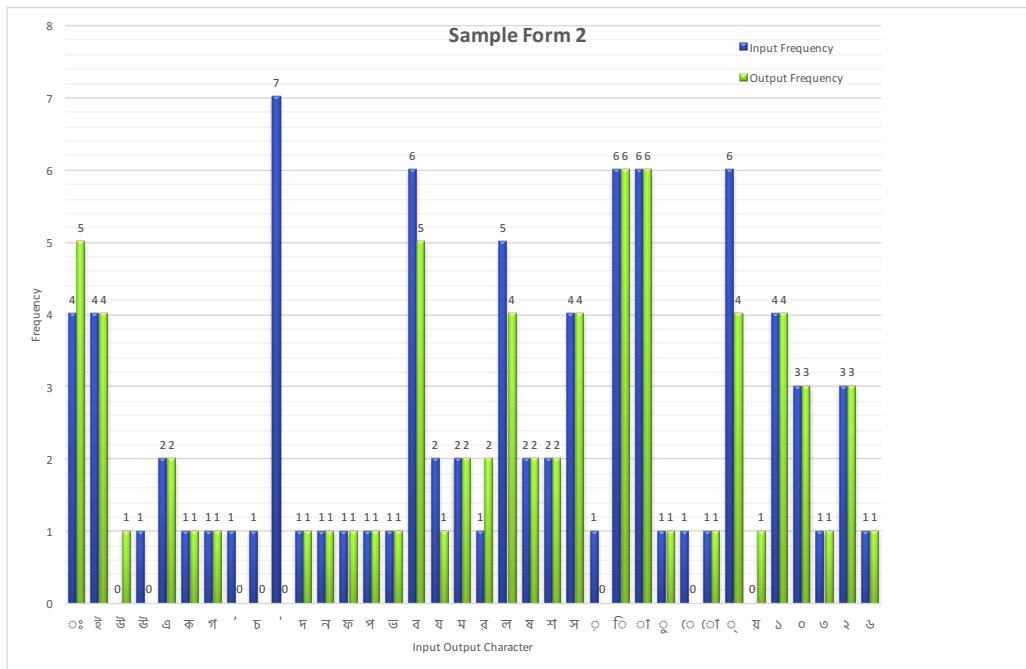


Figure 5.16: Bar chart Input Output Frequency of Sample form 2

5.3.3 Sample form 3

নামঃ	মাহবুউল আলম হাবিব
রোলঃ	১২১০১০৩৪
বিভাগঃ	সিএসই
বিশ্ববিদ্যালয়ঃ	ইউএপি
শিক্ষাবর্ষঃ	২০১৬

Figure 5.17: Sample Bangla form 3

Character	Input Frequency	Output Frequency
ং	4	5
ই	1	2
আ	1	1
উ	1	2
ঁ	1	0
এ	2	2
ক	1	1
গ	1	1
,	1	0
চ	1	0
'	8	0
দ	1	1
ন	1	1
প	1	1
ভ	1	1
ব	9	8
য	2	1
ম	3	3
০	1	0
ল	6	4
ষ	2	2
শ	2	2
হ	2	2
৮	1	0
ঁ	1	0
ি	7	7
ঠ	6	6
ঢ	1	1
ঠ	1	1
ঁ	6	4
ঁ	1	2
স	0	2
ঁ	0	1
ঁ	4	4
০	2	3
৩	1	1
২	2	2
৪	0	1
৬	1	1

Figure 5.18: Comparison between Input and Output frequency of Sample Input 3

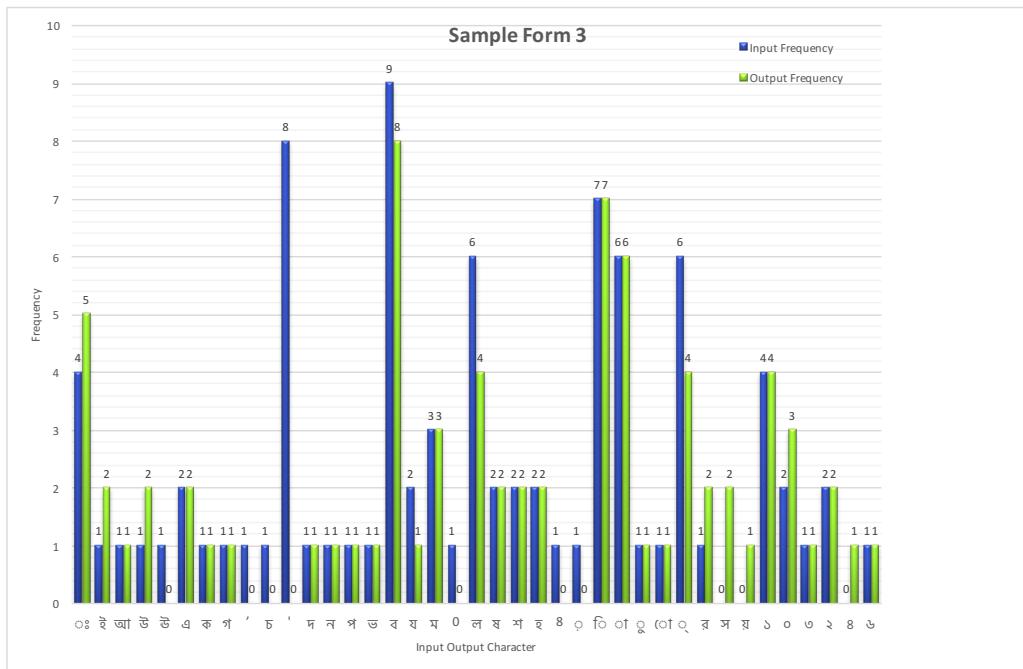


Figure 5.19: Bar chart Input Output Frequency of Sample form 3

5.3.4 Sample form 4

নামঃ	আল নোমান
রোলঃ	১২১০১০৬৩
বিভাগঃ	সিএসই
বিশ্ববিদ্যালয়ঃ	ইউএপি
শিক্ষাবর্ষঃ	২০১৬

Figure 5.20: Sample Bangla form 4

Character	Input Frequency	Output Frequency
ং	4	5
ই	1	2
আ	1	1
উ	0	1
ঁ	1	0
এ	2	2
ক	1	1
গ	1	1
,	1	0
চ	1	0
'	8	0
দ	1	1
ন	3	3
প	1	1
ভ	1	1
ব	6	5
য	2	1
ম	2	2
র	1	2
ল	5	3
ষ	2	2
শ	2	2
স	0	2
ঃ	1	0
ি	6	6
ঠ	6	5
ঁ	1	0
ঁঠ	1	2
ঁ	6	4
ঁয	0	1
ঁ	4	4
ঁো	3	3
ঁ	1	1
ঁ	2	2
ঁ	2	2

Figure 5.21: Comparison between Input and Output frequency of Sample Input 4

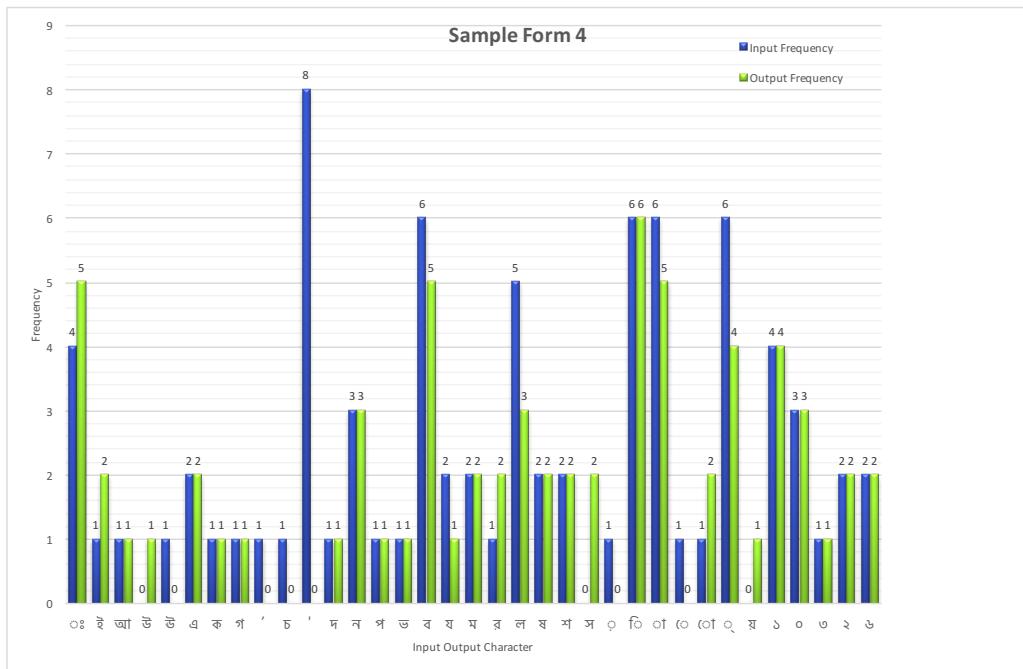


Figure 5.22: Bar chart Input Output Frequency of Sample form 4

5.3.5 Sample form 5

নামঃ	এহসুন আহমেদ তম্ময়
রোলঃ	১২১০১০২৯
বিভাগঃ	সিএসই
বিশ্ববিদ্যালয়ঃ	ইউএপি
শিক্ষাবর্ষঃ	২০১৬

Figure 5.23: Sample Bangla form 5

Character	Input Frequency	Output Frequency
ঃ	5	5
০	1	0
ু	1	2
আ	1	1
উ	1	1
এ	3	3
ক	1	1
গ	1	1
,	1	0
চ	1	0
ত	1	1
'	7	0
দ	2	2
ন	4	3
প	1	1
ভ	1	1
ব	6	5
য	3	1
ম	4	3
র	1	2
ল	4	2
ষ	2	2
শ	2	2
হ	2	2
স	1	3
ং	2	0
়	6	6
ঠ	5	4
ঙ	1	1
ঝ	1	1
ঢ	1	1
ঽ	7	5
ঘ	0	2
ঁ	4	4
ঁ	2	3
ঁ	3	3
ঁ	1	1
ঁ	1	1

Figure 5.24: Comparison between Input and Output frequency of Sample Input 5

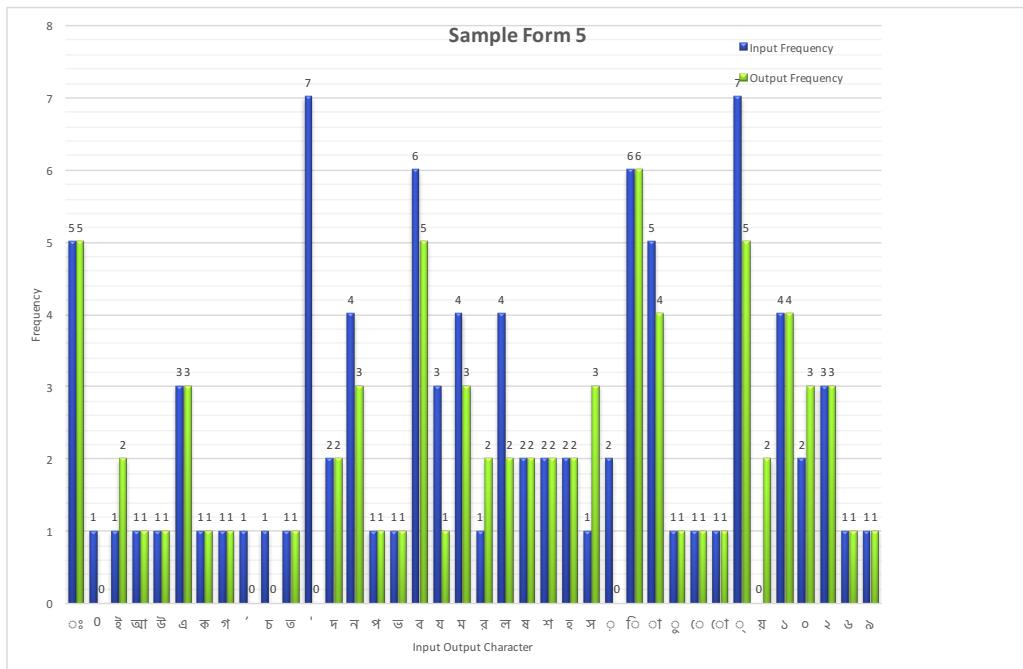


Figure 5.25: Bar chart Input Output Frequency of Sample form 5

According to these output of bangla from, we can say that here accuracy of our bangla training data upto 75%. In our training data big problem is with complex bangla word. Our trained tesseract can't detect complex bangla word perfectly.

Test - পরীক্ষা - test.

Chapter 6

Conclusion

This chapter summarizes the work presented in this thesis and figure out the advantages and limitations of the proposed algorithm. It also discussed about the scope of future works related to the proposed algorithm.

6.1 Discussion

From our experiments, we see that our methods could achieve the image segmentation purpose. For simple images with just a little texture inside, the result is quite good, and this method could also performs well on natural and landscape images. The above tests are based on the luminance or RGB based similarity measurement. When we test our algorithm on the texture images with the same similarity measurement, we found the result become worse. And the main reason is that the pixel value variance is really large in the texture region, which means pixels in the same texture region may have small similarities and are segmented into different groups.

This problem could be solved by using text on based similarity measurement, while the boundary position between segments are not so accurate determined due to the averaging operation by the sliding window. We knew that estimating the region properties is really an importance process. The Luminance-based method performs well on simple images with just a little texture inside.

And there's bit of noise with the output. We've tried to reduce noise as much as possible but it's the particular area that we've still to work a lot.

Character recognition techniques associate a symbolic identity with the image of character. Character recognition is commonly referred to as optical character recognition (OCR), as it deals with the recognition of optically processed characters.

Optical character recognition has many different practical applications. As observed from the experimental results, Tesseract OCR engine fares reasonably with respect to the core recognition accuracy on user-specific handwritten samples of isolated/free-flow text. Developing high accuracy, multi-font language data for robust, end-to-end processing for Tesseract was not within the scope of this study.

6.2 Future Works

There are still some things we can do for future works. At first, we will improve the stability of program. We want to modify the code of function to let programs more stable. Secondly, it's a chance to get a better adaptive method for image segmentation even that we have an adaptive method already. The third one is to generate the post-processing mechanism for region merging. We can write a code about merging groups with the same texture into a single group.

As the computer technology develops new methods for character recognition are still expected to be appeared and hope that new methods will decrease the computational restrictions. For the recognition of joined and split characters integration of segmentation and contextual analysis can be improve. An important future work can be implemented, is to improve the training process to be able to use real data for training instead of just synthetic data with character bounding boxes. This will greatly help to improve the accuracy on Bengali handwritten forms.

Our work outputs multi-page TIFF format images and corresponding bounding box coordinates in the Tesseract training data format. So, to achieve more decent accuracy we need more training data. By adding more training data we'll able to improve train data frequency, which we want do in near future.

Each and every aspects of our thesis can be improved upon. We just used our techniques to get a decent results but every techniques can be modified to get better result. For our thesis work, we just combining our techniques all together in order to achieve a decent result. There are many changes we would like to experiment with in the future.

Bibliography

- [1] Mr Santosh, LM Jenila Livingston, et al. Text detection from documented image using image segmentation. In *International Journal of Engineering Research and Technology*, volume 2. Citeseer, 2013.
- [2] Stephen Johnson. *Stephen Johnson on digital photography*. O'Reilly Media, Inc., 2006.
- [3] Richard A Haddad and Ali N Akansu. A class of fast gaussian binomial filters for speech and image processing. *Signal Processing, IEEE Transactions on*, 39(3):723–727, 1991.
- [4] Yudong Zhang and Lenan Wu. Optimal multi-level thresholding based on maximum tsallis entropy via an artificial bee colony approach. *Entropy*, 13(4):841–859, 2011.
- [5] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.
- [6] OpenCV. Canny edge detection in opencv. http://docs.opencv.org/trunk/d4/d22/tutorial_py_canny.html.
- [7] Ray Smith. An overview of the tesseract ocr engine. In *icdar*, pages 629–633. IEEE, 2007.
- [8] Tesseract. Documentation. <https://github.com/tesseract-ocr/tesseract/wiki/Documentation>.
- [9] Rafael C Gonzalez and Richard E Woods. Digital image processing. addision-wesley. *Reading*, 1992.
- [10] Roger D Boyle and Richard C Thomas. *Computer vision: A first course*. Blackwell Scientific Publications, Ltd., 1988.

- [11] OpenCV. Canny edge detector. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.
- [12] OpenCV. Contours. http://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html.
- [13] Claudio Rosito Jung and Rodrigo Schramm. Rectangle detection based on a windowed hough transform. In *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, pages 113–120. IEEE, 2004.
- [14] Chirag Patel, Atul Patel, and Dharmendra Patel. Optical character recognition by open source ocr tool tesseract: A case study. *International Journal of Computer Applications*, 55(10), 2012.
- [15] Tesseract. Trainingtesseract. <https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract>.
- [16] Md Abul Hasnat, Muttakinur Rahman Chowdhury, and Mumit Khan. An open source tesseract based optical character recognizer for bangla script. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 671–675. IEEE, 2009.
- [17] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* ” O'Reilly Media, Inc.”, 2008.
- [18] Sravan Ch, Shivanku Mahna, and Nirbhay Kashyap. Optical character recognition on handheld devices. *International Journal of Computer Applications*, 115(22), 2015.
- [19] Training tesseract. <https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract>.