5-1-2013

Skip Trie Matching for Real-Time OCR Output Error Corrrection on Smartphones

Aditya Vanka
Utah State University

Skip Trie Matching for Real-Time OCR Output Error Correction on Smartphones

By

Aditya vanka

A report submitted in partial fulfillment

of the requirements for the degree

of

MASTER OF SCENCE

in

Computer Science

Approved:

_____	_____
Dr. Vladimir A. Kulyukin	Dr. Daniel W. Watson
Major Professor	Committee Member

_____

Dr. Nicholas Flann

Committee Member

UTAH STATE UNIVERSITY

Logan, Utah

2013

ABSTRACT

Skip Trie Matching for Real-Time OCR Output Error Correction on Smartphones

by

Aditya Vanka

Utah State University, 2013

Major Professor: Dr. Vladimir A. Kulyukin
Department: Computer Science

Many Visually Impaired individuals are managing their daily activities with the help of smartphones. While there are many vision-based mobile applications to identify products, there is a relative dearth of applications for extracting useful nutrition information. In this report, we study the performance of existing OCR systems available for the Android platform, and choose the best to extract the nutrition facts information from U.S grocery store packages. We then provide approaches to improve the results of text strings produced by the Tesseract OCR engine on image segments of nutrition tables automatically extracted by an Android 2.3.6 smartphone application using real-time video streams of grocery products. We also present an algorithm, called Skip Trie Matching (STM), for real-time OCR output error correction on smartphones. The algorithm's performance is compared with Apache Lucene's spell checker. Our evaluation indicates that the average run time of the STM algorithm is lower than Lucene's.

(68 pages)

ACKNOWLEDGMENTS

I am grateful to my advisor, Dr. Vladimir A. Kulyukin, for his time and encouragement needed to complete this project. His recommendations and suggestions have been invaluable for both me and this project.

I would like to thank my committee members, Dr. Daniel W. Watson and Dr. Nicholas Flann for their support and valuable suggestions; they have been invaluable. I would also like to thank Dr. Stephan Clyde, President MDSC, for supporting our research in part.

I would like to thank my parents Sarat Chandra and Varalakshmi, my sister, Bhavani, my brother-in-law, Srikrishna, for their unconditional love and support.

Finally I would like to thank my friends Aditya, Ajay, Harish, Ramakanth and all my fellow students for sharing their enthusiasm and for comments on my work.

Aditya Vanka

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

According to a report by Prevent Blindness America and the National Eye Institute, Visual impairment and blindness affect more than an estimated 3.6 million adults in the U.S. and more than 1.4 million people are legally blind [1]. Assistive Technology (AT) is required to support the Visually Impaired (VI) shoppers in carrying out fundamental activities in their daily living, although most of these activities happen indoors, some regular tasks such as shopping happen outdoors. A research by Helal et al. in [2] explains the five steps involved in VI grocery shopping, which are travelling to the store, shopping for the desired product, making a payment, leaving the store, and travelling back home.

VI individuals may be able to get to a grocery outlet on their own using public transportation or with the help of guide dogs. However, shopping independently is a difficult task for the VI as they have to rely on some external help such as a sighted guide. Research in the field of assistive shopping systems has shown that such systems will help the VI shoppers to shop independently [3]. Assistive shopping systems exist that help VI shoppers to navigate the grocery store and search the products they want. Some examples of such systems are GorZi [6], RoboCart [4], ShopTalk [5], and ShopMobile [10]. These systems rely on customized hardware and also customized store environments. Customized hardware has two problems: first it is costly to produce such devices, and second its maintenance is difficult. Grocery store owners are reluctant to customize their store environments because this adds to the cost of maintaining the store. ShopMobile1 [10] and ShopMobile2 [11] systems address the limitations of the

customized assistive shopping systems. ShopMobile2 replaces a dedicated barcode scanner by allowing the VI shopper to scan barcodes on products and shelves quickly and reliably using only the smartphone, which will be discussed more thoroughly in Section 2.1.

Most of the assistive systems usually help VI shoppers to find the product they are looking for by identifying the product using a barcode and then giving out a verbal description of product matching that barcode. This approach works acceptably as we can see from the experiments using ShopTalk, ShopMobile1, and ShopMobile2 systems. The ShopMobile2 system also proves that with the application of computer vision concepts on high-end smartphones, quality assistive shopping systems can be built. Once a VI shopper knows the product, the next important pieces of information about the product are the nutritional facts and the ingredient information. Nutrition facts information is important to the VI shopper to proactively manage their diets and to prevent illnesses that occur because of mismanaged diets. Knowledge of allergen information can help the VI shoppers make a choice whether to buy a product or not.

The Specific Aims of this project are to:

1) Study the performance of existing OCR systems available for Android platform and choose the best to extract the nutrition facts information from a grocery store item;
2) OCR is an expensive process; since image processing is also computationally intensive we try to minimize the pre-processing applied to images before the OCR process. We rely on pictures captured from an Android cell phone camera in a grocery store environment. This brings in some noise to the images because of

various reasons such as uneven lighting, bright spots (due to overhead focus lights) etc. So, we apply minimal and fast image processing techniques to improve the recognition rates of OCR system and study their effect on the recognition rates. Specifically, we are going to study the impact of different image binarization techniques on OCR output;

3) Spelling errors are common in the output of OCR, to enhance the results of the OCR process we try to post-process the output of OCR by applying spelling correction algorithms. Since OCR is an expensive process, minimizing the time spent on post processing is important. So, for processing the OCR recognized text we developed a fast spelling correction algorithm called SkipTrie Matching (STM). We also use the spelling correction algorithms available in an open source framework called Apache Lucene [33]. We implement and compare the performance of STM to that of Lucene.

This project marks the first step toward building a real-time mobile Nutrition Management System to aid the VI shoppers. An overview of flow of control between different modules in our application is illustrated in the Figure 1.

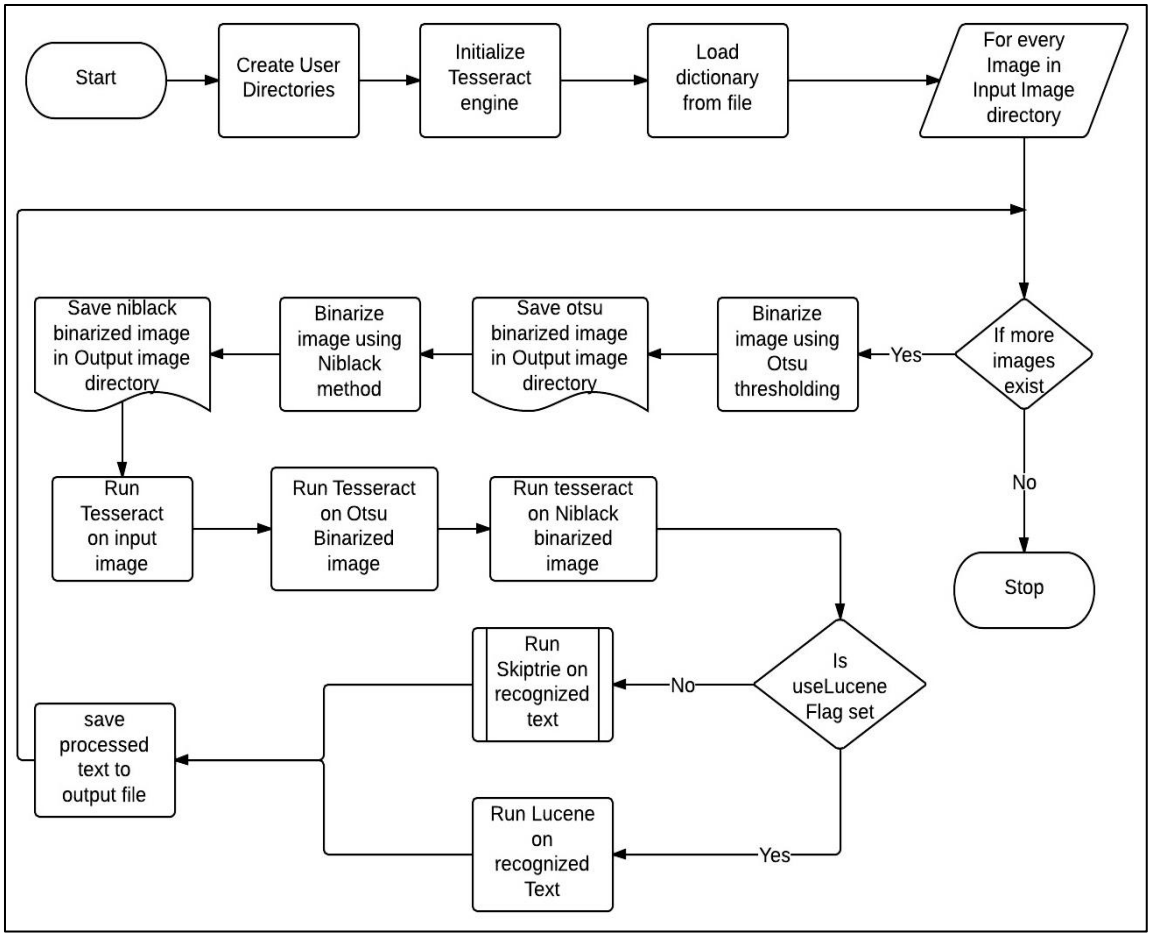


Figure 1. Flow of Control Between different Modules

The report is organized as follows: Chapter 2 presents an overview of the related work done on the assistive technology systems. Chapter 3 evaluates two open source OCR Systems on an Android device. Chapter 4 provides an overview of different stages involved in an OCR process. Chapter 5 presents the task analysis and discusses in detail about the implementation of the preprocessing techniques used and the SkipTrie Matching algorithm. Chapter 6 presents an overview of the system design of the Android application we develop and also provides details of the implementation. Chapter 7 discusses the results of the experiments using the implementation described in Chapter 6. Chapter 8 presents the future work and ends with a conclusion in Chapter 9.

CHAPTER 2

RELATED WORK

Several Assistive technology systems have been developed to aid the VI in independent grocery shopping. The Computer Science Assistive Technology Laboratory (CSATL) at Utah State University (USU) has developed four assistive technology systems such as RoboCart [4], ShopTalk [5], ShopMobile1 [10], and ShopMobile2 [11]. Other accessible shopping systems like GorZi [6], Trinetra [8] [9], and iCare [7] have also been developed at various other institutes. Most of the earlier systems required specialized hardware and changes to store environment, but advances in mobile computing have paved a way for systems like ShopMobile2, which completely rely on using mobile computing to provide assistive technology systems. A brief Overview of some of these systems is presented in the reminder of the chapter.

2.1 ShopTalk

ShopTalk was the second assistive technology system built at CSATL. This system used a wearable small-scale computing system. It used an OQO wearable computer, a Belkin keypad attached to a backpack, and a handheld barcode scanner. In addition, it used Modified Plessey (MSI) shelf barcodes for both store navigation and product search. A Barcode Connectivity Matrix (BCM) was generated to map the store using verbal instructions that were generated to guide the VI shopper. This system relied on the computation of BCM, which required access to a product database of a grocery store, which is not a feasible solution [5].

Figure 2. Hardware Components of ShopTalk

2.2 ShopMobile

The ShopMobile system was a smartphone based barcode scanner. A camera-abled smartphone in a case was placed on the lip of the shelves with the help of stabilizers. With the help of a barcode reading computer vision application on the smartphone, the system read the shelf barcodes and read out the product details via an over-the-ear head piece with the help of screen reader software [10].

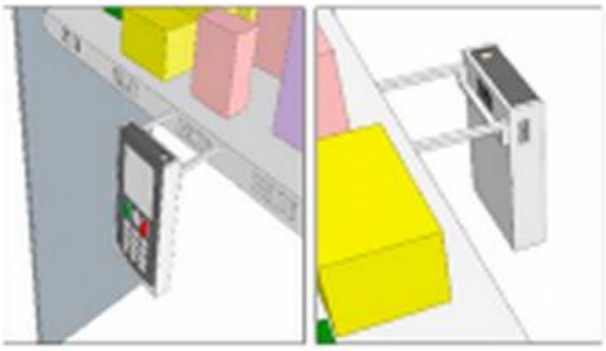


Figure 3. Hardware Components of ShopMobile 1 [10]

2.3 GroZi

GroZi [6] is the assistive shopping project designed at UCSD. GroZi uses three components for its operations. It has an accessible web site for blind and VI users to create shopping lists; computer vision software to scan for products; and portable devices that can give users verbal feedback by executing computer vision algorithms. If the system detects any product in the video stream that is on the shopping list it guides the user to get that item by using object recognition algorithms. The GroZi's long-term objective is to enable the user to sweep the portable camera's field of view across the grocery shelves. GroZi has two kinds of images: *in vitro* and *in situ*; one for training and the other for testing. The *in vitro* images are images of products taken under ideal lighting and perspective conditions. The *in situ* images are obtained from actual video streams in the store [6]. This system also requires specialized hardware.

2.4 iCARE

iCARE [7] is the an assistive shopping system that was designed at Arizona State University (ASU). This system is assumed to provide indoor navigation, contents in each section of the shopping area, and a user interface for querying product databases. The design of this system includes a PDA with Bluetooth, Wi-Fi, a screen reader, and an RFID reader embedded in a glove. The RFID reader in the hand glove reads IDs from the products, looks them up in a store database through a Wi-Fi connection, and gives instructions to the user enabling him/her to navigate and shop at the same time. The best feature of the system is "Browsing." Upon waving the hand glove across the shelf, the PDA will deliver messages such as "passing dairy section" or "passing coffee section." When the user is inspecting a product, the user will receive the individual package's

price, weight, ingredients, and nutritional data presumably from the RFID tag read from the package [7]. This system assumes a lot of information to be available from the store, which makes it an infeasible solution.

2.5 ShopMobile2

ShopMobile2 [11] [12] is an enhancement over ShopMobile1. It is the latest assistive shopping system developed at CSATL and is based on computer vision algorithms implemented on an Android smartphone. This application provides the VI shopper with an eyes-free barcode scanner, a tele-assistance module, and an OCR module. The MSI and UPC barcodes are decoded by the eyes-free barcode scanner. The OCR module helps the users to get information about the Nutritional Facts Table (NFT) on the product. The tele-assistance module helps the VI shoppers by transmitting a video stream to a remote assistant who can give instructions to VI about the product. The advantage of this system is that it runs on a VI shopper's mobile phone and does not require any additional hardware installations in the shopping centers. This solution aims to provide a truly independent shopping experience to the VI shopper. The advantage of this system is that it does not require any modifications to the store environment.

Figure 4**. Hardware Component of ShopMobile2**

CHAPTER 3

OCR ENGINE EVALUATION

Most of the previous research at CSATL was concentrated on product identification through various systems such as ShopTalk and ShopMobile2. While ShopMobile2 reliably proves that powerful smartphones can be efficiently used to identify products using MSI barcodes and UPC barcodes, after a product is identified the next important pieces of information are the nutrition facts and allergens disclosure available on a grocery product. To decode the nutrition facts and/or allergy information from the product image, an Optical Character Recognition (OCR) system is required. Since the main aim of this project is to study and implement techniques to improve the recognition rates of the Optical Character Recognition Engine by targeting the pre-processing stage and the post-processing stage of the OCR process (more on this in Section 4.1), we first try to choose the best OCR Engine available for the Android platform. This chapter details the approach we have taken to choose an OCR Engine.

3.1 Introduction to OCR

Optical Character Recognition can be defined as the machine recognition of text in an image. Modern OCR systems can recognize many different fonts, as well as typewriter and computer-printed characters. Advanced OCR systems can even recognize hand printing. The very first attempts of OCR date back to 1870 when a device called retina scanner was invented by C.R. Carey of Boston, which was built using an array of photocells and was used to transmit images. Twenty years after that, sequential scanner was invented by Polish P. Nipkov. During the initial period of the 20th century, attempts were made to develop devices to aid the blind with OCR. During the 1950s, the first OCR

machines were commercially available and the first OCR machine was installed at Reader's Digest that was used to convert sales reports to electronic format to be fed into a computer. The first generation OCR systems were only able to read specific letter shapes. During the 1970s, OCR systems matured a little and were able to read machine printed and hand printed characters. After a comprehensive study of requirements for OCR, an American standard OCR character set was defined with a highly stylized font [14].

OCR systems have become quite robust over time and are now widely available on personal computers. However, a recent boom in the use of powerful smartphones has shifted the focus into developing OCR systems for wearable computers (smartphones). Because the processing capabilities of the mobile devices have seen a radical improvement over the last few years, resource hungry OCR processes can be acceptably performed on a mobile device. Recent years have seen a lot of research on building mobile systems for text extraction both in the commercial space and in academia. Commercially, ABBYY [14] offers a powerful but compact mobile OCR engine that can be integrated within various mobile platforms such as iOS, Android, and Windows mobile platforms. Another company called WINTONE [15] claims that its mobile OCR software can achieve recognition accuracy to the tune of 95 percent for English documents. In academia, a project in [16] implemented a business card reader in mobile devices with a built-in camera. As it is apparent, most of these systems only serve commercial interests.

The idea of using a wearable computer, and making use of its inbuilt camera to perform OCR, opens up a great deal of applications for VI shoppers. A few OCR systems to aid VI shoppers were developed, such as those that read bank notes [17] and signs

[18]. Making use of OCR systems to read out the contents of nutrition facts on a grocery item to VI shoppers will make the shopping experience feel more independent.

3.2 Evaluation of OCR Engines for Android Platform

Significant research and time has been spent on the development of some of the open source OCR systems available today. We attempted to find the best available Open Source OCR system and use it in this project. Some of the open source OCR systems that are available today for PC's have been ported to be used on today's mobile platforms like Android and iOS. Examples of such systems are Tesseract [19] and GOCR [20]. The OCR process tends to be resource intensive, and image quality plays a major role in the amount of text recognized and the accuracy of the recognized text. Although the pictures captured by smartphone cameras have decent resolutions, images tend to be skewed and have varying illumination.

Here we will look at how both Tesseract and GOCR frameworks work, then we will try to determine the performance these OCR engines on the Android platform by performing OCR on a sample set of nutrition facts table sub-images captured from random grocery items using an Android smartphone.

3.2.1 Tesseract OCR Engine

Tesseract is an open source OCR engine developed by HP between 1985 and 1995. It came to limelight when it was rated highly at The Fourth Annual Test of OCR Accuracy held in 1995 at the University of Nevada, Las Vegas' Information Science Research Institute. After years of dormancy, HP released Tesseract to the open source development community in 2005. Later it was adopted as a Google project [19] and has since seen significant development.

Tesseract processes an image in steps [21], and each step has important phases in the pipeline. The first phase is the segmentation and the second phase is recognition. In the first step a connected component analysis is performed and the outlines of the components are stored and blobs are created by grouping the outlines. The second step involves organizing the blobs into text lines that are analyzed for proportional text or fixed pitch text. The third step breaks the lines into words by following the spacing between characters. In the fourth step character cells are identified for both fixed pitch and proportional text.

Tesseract's recognition happens in two passes. In the first pass it tries to recognize each word, and words with satisfactory confidence are passed to train an adaptive classifier that helps to recognize text in the next lines of the image more accurately. In the second pass, words recognized with lesser confidence in the first pass are subjected to another recognition phase to see if they can be better recognized with the help of a trained adaptive classifier. Feature analysis is used to determine the characters in a word [21].

3.2.2 GOCR OCR Engine

GOCR is an OCR (Optical Character Recognition) program, developed under the GNU Public License. Its development started in 2005 and the project has been largely inactive for the last few years. However, this is one of the OCR engines that was ported to the Android platform, so we decided to try to measure its performance against the Tesseract OCR engine. GOCR only works on binarized and non-skewed images.

GOCR's architecture processes an image in several steps. The first step preprocesses the image, which involves several sub-steps such as box-detection, zoning, and line detection. The second step calls the OCR engine. There are three OCR engines in GOCR

and two of them are experimental. The second step is repeated for characters that are not recognized in the first pass. The third step does post processing to identify the text type. GOCR relies on the character detection by character pixel pattern analysis [20].

3.2.3 Experiment Results

To compare OCR engines we use two important parameters: speed and accuracy. Speed is important in order to give the VI user a quick feedback about the contents of the image. Accuracy on the other hand determines the quality of feedback provided to the user. If the accuracy of the content recognized is good, the VI can comprehend the contents better. If the accuracy of the recognized text is not very good, then the VI has to concentrate more to comprehend the contents and sometimes they may not be able to assimilate it. There has to be a certain degree of tradeoff between the speed and accuracy of the recognized text to provide timely and qualitative feedback to the user. We developed an Android application that reads a database of 200 images on an SD Card, and processes each one of the images using both Tesseract and GOCR. These images are manually cropped lines from a nutrition facts table. An example of such images can be found in Figure 5**.** We have tabulated the results based on both accuracy and processing time. The following tables show the results of a comparison of accuracy and speed for both of the frameworks.

Figure 5. Line Images Extracted From a NFT

3.2.3.1 Accuracy

To analyze the accuracy of recognized text, we divided the text recognized by both frameworks into three categories, namely Complete, Partial, and Garbled/Unidentified. The Complete category includes the count of those images from which the exact text was decoded by the OCR system. The Partial category includes the count of those images from which most of the text was identified, but where some of the characters were either missing or substituted with other characters. The Garbled/Unidentified category represents the count of those images from which none of the text is decode or groups of characters unrelated to the text contained in the image were reported as recognized text.

Table 1. Comparision of Tesseract and GOCR Accuracy

	Complete	Partial	Garbled/ Unidentified
Tesseract on Device	146(73%)	36(18%)	18(9%)
GOCR on Device	42(21%)	23(11.5%)	135(67.5%)
Tesseract on Server	158(79%)	23(11.5%)	19(9.5%)
GOCR on Server	58(28.99%)	56(28%)	90(45%)

Looking at the results in Table 1, it is apparent that the Tesseract framework does a better job of extracting text from most of the images. The rates of recognition of the GOCR system indicate that most of the time the text extracted is either garbled or unidentifiable. Ninety-one percent of the time, Tesseract is able to extract understandable text from images when run on a device, whereas GOCR is only able to extract understandable text from 32.5% of images.

3.2.3.2 Processing Time

To evaluate the processing times taken by each of the frameworks, we ran the above described experiment five times for both Tesseract and GOCR. The processing times were then tabulated as shown in Table 2**.** All of the times indicated are in milliseconds. In Table 2, The AVG/Sample is the average processing time per entire sample, and the AVG/Image is the average processing time per image**.**

Table 2. Comparision of Running Times of Tesseract and GOCR

	Run 1	Run 2	Run 3	Run 4	Run 5	AVG/Sample	AVG/Image
Tesseract on Device	128238	101438	101643	109678	103205	110439.6	552.1
GOCR on Device	50349	47746	48964	52450	48247	49019.6	245
Tesseract on Server	38958	38061	37850	9891	39032	38289.6	191
GOCR on Server	21253	20842	20195	21182	20520	20763.3	103.8

The first observation that is clear from the data obtained is that there is no significant difference in the processing times of the OCR process on images across multiple runs. Second, Tesseract takes longer to run than the GOCR. This difference can be attributed to the amount of text recognized by each of the frameworks. Since GOCR extracts very little information it runs much faster. Tesseract on the other hand, extracted information from most of the images, taking more time to process. On an average, Tesseract takes 552.1 milliseconds to process one image, whereas GOCR takes 245 milliseconds to process an image.

From the above experiment we determine that Tesseract is the slower of the two, but only Tesseract produces text that is comprehensible. Because of this, we choose the Tesseract framework to be the better of the two considered frameworks for an OCR system on Android smartphones. Going forward, whenever we use the term OCR Engine it can be treated synonymous to the Tesseract OCR Engine.

CHAPTER 4

STAGES OF OCR PROCESSING

This chapter first explains the general processing steps involved in an OCR application. Second, it details an approach used in Localization and Segmentation step of OCR process to localize and segment a NFT. Third, it details of the image processing techniques in the preprocessing stage to increase the recognition rates of OCR Engine.

Four our study, we relied on pictures captured from an Android cell phone camera in a grocery store environment, which brought some noise into the images due to reasons such as uneven lighting and bright spots due to overhead focus lights.

4.1 Stages of OCR Processing—An Overview

An OCR Process consists of several processing stages [22]. Figure 6 illustrates the common stages involved, including Capture Image, Localization & Segmentation, Pre-Processing, Recognition, and Post Processing. We will review each stage of the OCR process in the following sections.

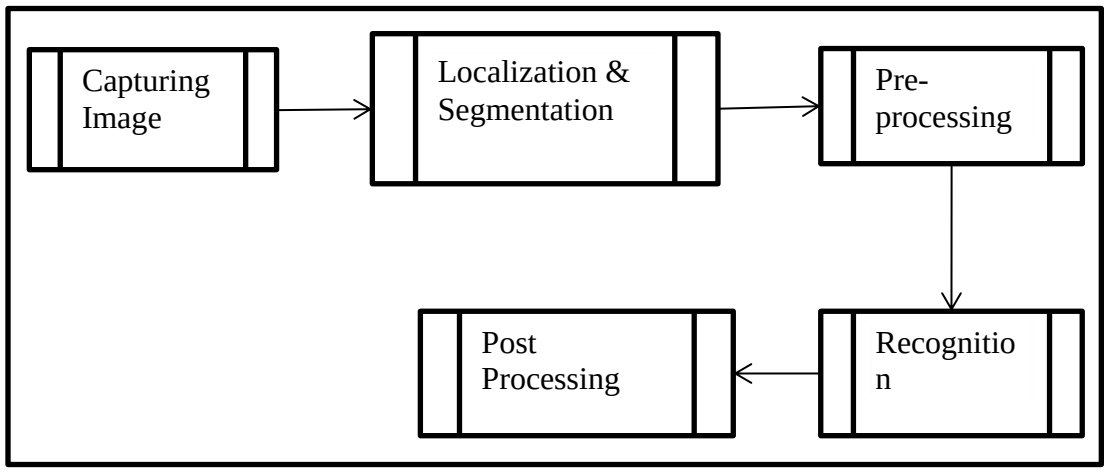


Figure 6. Stages of OCR Processing

4.2 Capture Image

Capturing the image is the first step in the process. The Interactive Camera Alignment Module (ICAM) [11] [12] of the ShopMobile2 application was built to aid the VI in capturing GroZi a picture of the grocery item without much distortion or skew. This ICAM module makes use of sensors commonly available on mobile devices such as the orientation sensor. This module reads the sensor data and helps the VI to adjust the camera based on specific verbal instructions such as “roll right” and “pitch down”. Figure 7 below illustrates the pitch and yaw planes. This module was developed to reduce the amount of pre-processing [23].

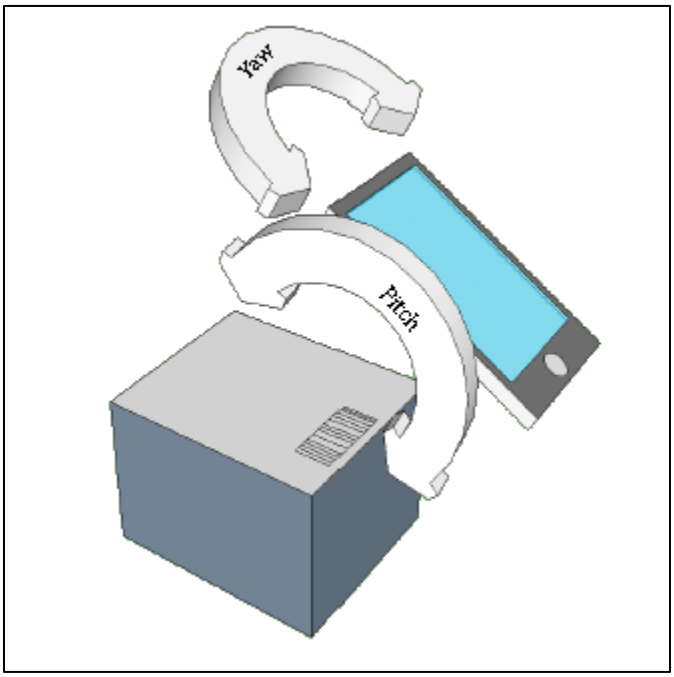


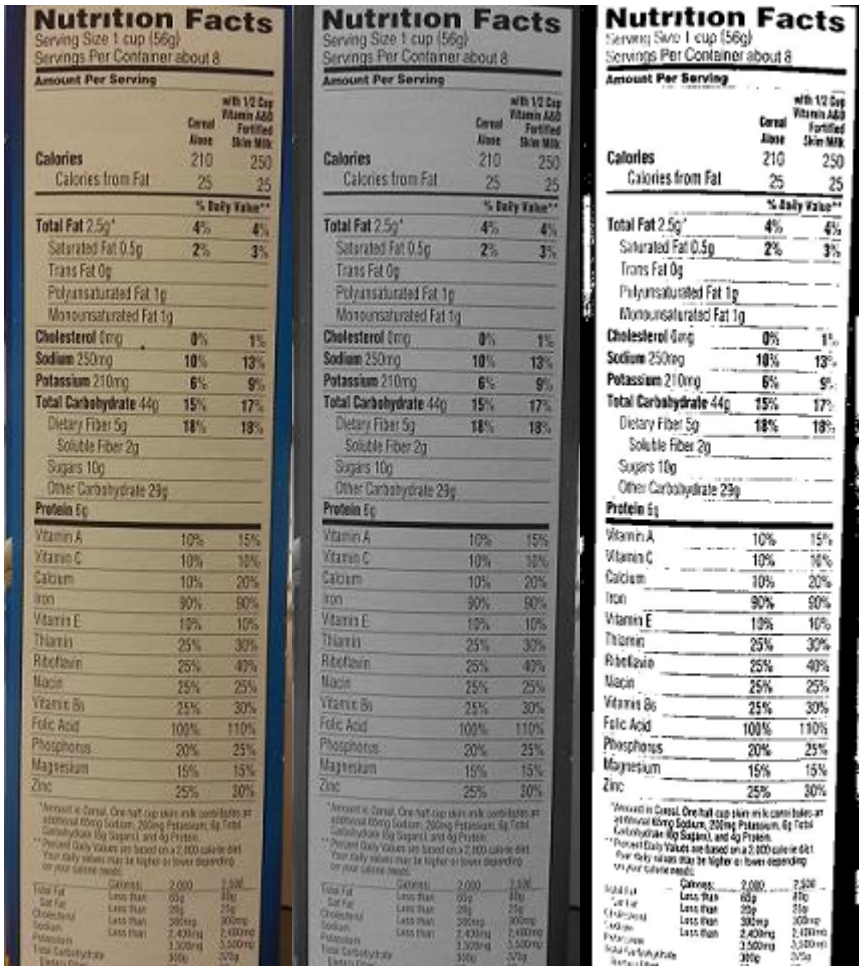
Figure 7. Pitch and Yaw Illustration [23]

4.3 Preprocessing

The image captured from the camera of a mobile device is susceptible to noise. Shadows are common while capturing images using a mobile phone camera, which leads to different illumination levels at different parts of the image. Also, in a grocery store environment, the image tends to have highlights due to bright overhead lights. Correcting these kinds of noise is essential to obtaining acceptable results during both localization and segmentation, as well as for OCR. A well-known technique to solve the illumination problem is by converting a multilevel RGB Figure 8(a) image (Figure 8(a)) into a binary image (Figure 8(c)). The OCR process does not gain on using a full RGB color image; binary images are entirely sufficient so there is no need to use more complicated and harder-to-process color images.

The multilevel RGB image is first captured and converted to a greyscale image (Figure 8(b)). A greyscale image has only varying shades (0-255 levels) of grey; in simple terms it is an image that only contains equal intensities of R, G, and B colors. Second, the greyscale image is subject to a process called binarization or thresholding, which converts the image from varying levels of grey to one with only two intensity levels (either 0 or 255).

The process of binarization is very simple; each pixel of the input image is compared to a threshold and if the pixel intensity is less than the threshold the value of the pixel is set to 0 or 255. Prior literature classifies the thresholding process into two main categories [24]: global binarization [25] [26] method and local binarization method [29] [30].



a) RGB Image b) Greyscale Image c) Binarized Image

Figure 8. RGB, Greyscale and binarized equivalents of a sample NFT image

The Global thresholding technique, such as Otsu's method, chooses a single threshold for the entire image. Each pixel is compared to the calculated threshold and chosen to be either a foreground pixel or a background pixel. These global thresholding methods are inexpensive and work well with images that have clearly defined foreground and background regions. However, these global thresholding techniques produce unwanted noise in cases where the intensities in different sections of the image happen to vary.

The Local thresholding technique, such as Niblack's method and the Sauvola's method, compute the threshold of each pixel based on the intensities of the group of pixels surrounding it. In local thresholding the image is divided into a local sub-image called a window, and thresholding is applied within that window. In the case of Niblack's and Sauvola's method, the local threshold is calculated using the mean and standard deviation of pixel values within a window. Local thresholding usually provides better results when compared to global thresholding techniques. The performance of these algorithms depends on the size of window.

Since different binarization methods produce different results for different kinds of images, we compare the performance of the OCR process (Tesseract) on line images extracted from NFT using different binarization methods in Section 7.2.

4.4 Localization and Segmentation

Localization

The third step in the OCR process involves the localization and segmentation of the NFT from the captured image. Significant research has taken place at CSATL while developing the ShopMobile2 application to localize and segment the nutrition facts table.

First, it is assumed that a complete nutrition facts table that is not cropped is in the captured image. The structure of a nutrition facts table is usually similar across different grocery items. There is a rectangular box surrounding the nutrition facts table delineating it from the rest of the image and different sections of the table are separated by horizontal lines. A common technique used in Object Localization is based on Horizontal and Vertical projections. The pixels in the image are first classified as foreground and background pixels. The Foreground pixels are those that represent the information an image is carrying. The background pixels are all other pixels that do not count as Foreground pixels. A Horizontal Projection (HP) is the number of foreground pixels in a row of an image. A Vertical Projection (VP) on the other hand is the number if foreground pixels in a column of an image. Using HP and VP we can find out our region of interest, which in our case is the NFT. The Localization of the NFT comprises of three steps.

Step 1

Approximate locations of vertical bounding lines of the NFT are found. In this step the input image is subjected to an efficient Horizontal Line Detection Kernel as discussed in [31]. This kernel produces an image as shown in Figure 9.

Step 2

The image from the above step is then subjected to vertical projections. The Figure 10 shows how the vertical projections look on the image. A threshold is calculated as the average of the number of foreground pixels Figure 9. The NFT area has foreground pixels greater than the threshold T as indicated by the red marker in Figure 10; based on this threshold the vertical bounding lines of the NFT are computed.

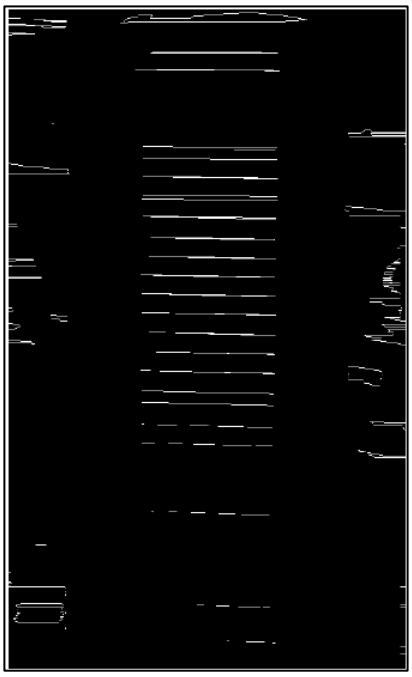


Figure 9. NFT image showing Horizontal Projections

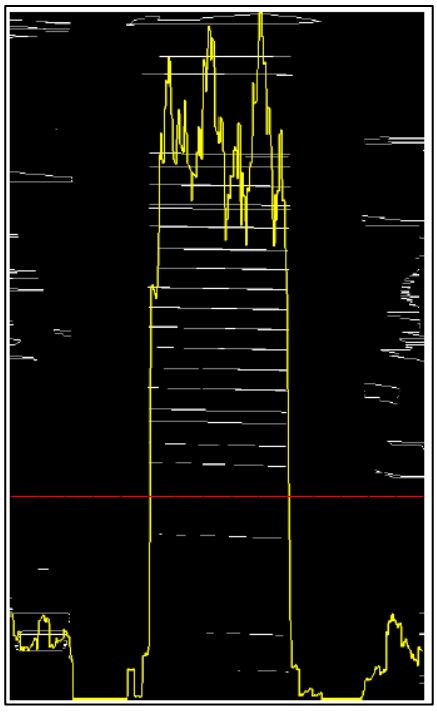


Figure 10. Image showing vertical projections with red Threshold marker

Step 3

In this step the upper and lower boundary lines of the NFT are computed. The image in Figure 9 is subjected to horizontal projections, a threshold and the horizontal bounding lines of NFT are calculated similar to the approach used in step 2.

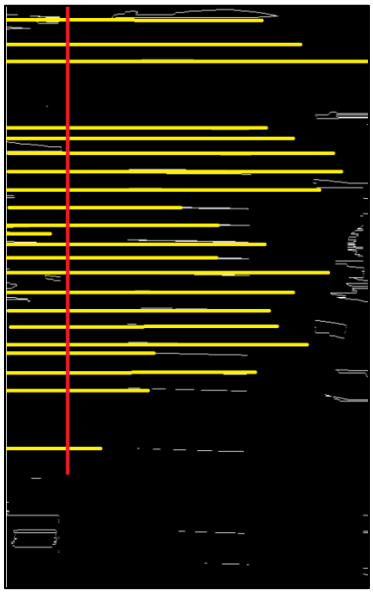


Figure 11. Image showing Horizontal projections and red Threshold line

Using the horizontal projections as shown in Figure 11, the NFT is localized. This process is explained in more detail in [11] application.

Segmentation

In [11][12], the process is described that segments each of the lines in the nutrition facts table in detail. The approach used for segmenting the NFT first finds the horizontal projections for the localized NFT image; it then finds a threshold of foreground pixels based on the Geometric mean of the projections. Using this threshold,

the algorithm differentiates lines and text regions and extracts the line images as shown in Figure 12.

Figure 12. A NFT with segmented Line Images

4.5 OCR Recognition & Post-Processing

This stage relies on the Tesseract engine to extract text out of the line images that we get from the Localization and Segmentation stage above. Please refer to the Section 3.2.1 for details about how the Tesseract OCR Engine works.

The text recognized by the OCR engines tends to have spelling errors. The final stage of OCR processing tries to correct the spelling errors to make the output of the OCR process more comprehensible to the VI shopper. A more detailed explanation about spelling correction and the general spelling correction algorithms is given in Section 5.3 of Chapter 5.

CHAPTER 5

TASK ANALYSIS AND INTRODUCTION TO SKIP TRIE MATCHING ALGORITHM

The main aim of this project is to study the techniques to improve the recognition rates of Tesseract by targeting the pre-processing stage, and to improve the quality text output of Tesseract during the post-processing stage of the OCR process. The first half of this chapter gives an overview of algorithms used during the pre-processing stage of OCR in our application. The second half focuses on the algorithms used in the post-processing stage for string correction. We also introduce a new Skip Trie spelling correction algorithm developed based on Trie data structure and discuss its working in detail.

5.1 Motivation

Localization, Segmentation, and Tesseract take up the largest chunk of time during the OCR process; we want to minimize the time spent on all other stages of OCR process. Since binarization has considerable impact on the output of Tesseract, we include it in the pre-processing stage. We choose not to perform additional pre-processing techniques such as scaling and smoothing, as they impact the overall time taken by the OCR process by either increasing the pre-processing time or by increasing the Tesseract processing time.

We developed an efficient spell correction algorithm called the SkipTrie Algorithm used during the post-processing phase in the OCR Process to correct the spelling errors in the output of Tesseract. Since very minimal configuration can be done on Tesseract we consider it as a black box.

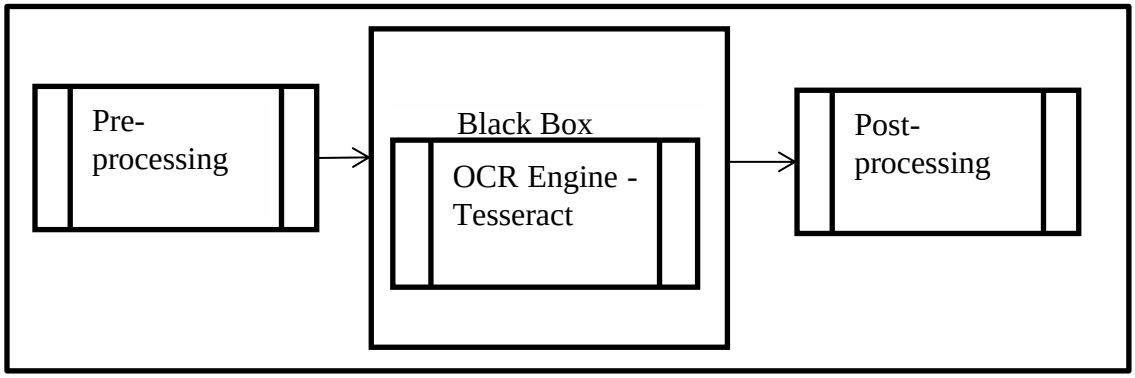


Figure 13. Critical sub-processes of an OCR System

5.2 Binarization Algorithms

We discussed some of the Image Binarization techniques in Section 4.2. Global thresholding techniques are not of much use in our case as we assume the presence of noise in the images captured by a mobile phone camera in a grocery store environment. For this project we used two local adaptive binarization techniques: Modified Niblack's binarization [31] and Modified Otsu algorithm [32] from the Leptonica image Processing Library bundled with Tesseract engine. The following section briefly explains both the modified Niblack's method as well as the modified Otsu method.

5.2.1 Modified Niblack Binarization Method

A modified Niblack Binarization algorithm [31] was developed at CSATL to be more efficient than the original Niblack's method. In the original Niblack's method, the image is first divided into an $n \times n$ pixel sub-image. For each pixel (x,y) in the sub-image a threshold $T(x, y)$ is calculated using the following equation:

$$T(x, y) = m(x, y) + k \times s(x, y),$$

Where $m(x,y)$ and $s(x,y)$ represent the mean and standard deviation of the $n \times n$ sub-image, respectively, and k is a user defined parameter. Since threshold is calculated for each and every sub-image, this original Niblack's method is computationally expensive. The modified Niblack's method slightly modifies the original method. The modified method calculates one threshold for an entire $n \times n$ sub-image. In the modified Niblack's method the equation used to compute the threshold $T(x,y)$ is as follows:

$$T(x,y) = \begin{cases} m(x,y) + k \times s(x,y) & \text{if } s(x,y) \geq S \\ T^c & \text{otherwise} \end{cases}$$

We used the parameters $T^c = 127$, $n = 50$, $S = 12.7$ and $k = -0.25$ in our application. These values were derived based on experimentation.

5.2.2 Modified Otsu Binarization Method

We have chosen the other binarization method as a modified Otsu method based on a study of its use in degraded images [33], this implementation is part of the Leptonica image processing library that bundles with the Tesseract OCR engine. In the standard Otsu method, different possible thresholds are used on a histogram of foreground and background pixel values. Threshold is then chosen so as to maximize the variance of the distributions of foreground and background pixels. A score function is maximized that is the product of the number of pixels on each side of the threshold times the separation of the mean values. This approach works better when the ratio of the number of background pixels to foreground pixels is not too large. If there are few foreground pixels, the threshold will be chosen well up the lower slope of the background distribution, resulting in many background pixels being identified as foreground.

For clean images where there is not a large variation in the background pixels standard, Otsu works quite well. Leptonica uses a modification of Otsu to minimize this problem. Instead of selecting the threshold value to be at the maximum of the score, it is chosen to be at the minimum histogram value such that the score is within some fraction of the maximum. In the modified Otsu method, the input image is divided into n × n sub-images and the Otsu threshold is determined separately for each sub-image. This modified Otsu method also has an optional smoothing operation on the sub-image thresholds.

From our experiments the modified Otsu method produces text that is evenly weighted and lighter when compared to that of the modified Niblack's method where the text tends to be heavier.

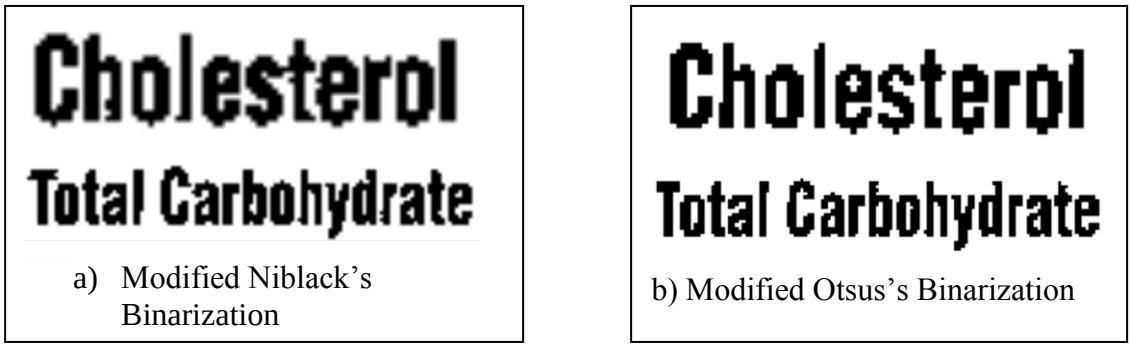


Figure 14. Niblack binarized images vs Otsu binarized images

5.3 Post Processing—Spell Checking Algorithms

Typically the text recognized by OCR engines tends to have spelling errors caused by confusion that arises when recognizing characters with similar features. For

example, in our experiments ‘t’ is often recognized as ‘l’ and ‘u’ is often recognized as ‘ll’ by Tesseract. Another type of error that might creep into OCR recognized text is related to deletion of character from a word. This can occur due to the OCR engine not being able to recognize a particular character of a word.

Spell Checking and spell correction have been an active research area since early 1960’s. A survey by [37] highlights the research done in the field of spell checking and correction, and it is a conclusive guide. According to [38] Spelling correction can be broadly classified into two different categories: 1) non-word errors and 2) real-word errors. A non-word error can be described as an error that occurs if the OCR recognized word is not contained in the target dictionary. A simple example of a non-word error is recognition of word by OCR as ‘polassium’ instead of ‘potassium,’ which is a dictionary word. A real-word error can be described as an error that occurs if the recognized word is correctly spelled but incorrectly used in a context. An example of real-word error is recognition of the words by OCR as ‘nutrition fats’ instead of ’nutrition facts.’ Here both the words ‘fats’ and ‘facts’ exist in the dictionary, but the context in which the word is recognized is erroneous.

A Large number of spelling errors in our experiment were non-word errors. Hence, we only look at techniques to correct these non-word errors. A common method for spell checking is to compare the recognized words with words in a target dictionary of lexicon. A word is declared a non-word if it is not in the dictionary. To correct the spelling, similar words are fetched from the dictionary based on similarity criteria. Two common approaches to establish similarity between words are 1) Edit Distance and 2) N-gram approach.

5.3.1 Edit Distance

The term edit distance was defined by Wagner [1974] as the minimum number of editing operations like insertions, deletions, and substitutions that are required to transform one string into another. There are two popular implementations of Edit Distance algorithms: Levenshtein distance [35] and Damerau-Levenshtein distance [36].

Levenshtein distance: The minimum cost sequence of single-character replacement, deletion or insertion operations that transforms the one string into another target string. For example the Levenshtein distance between the words ‘Galriius’ and ‘Calories’ is 4.

Damerau-Levenshtein distance: Supports all of the operations from the Levenshtein distance and further allows transposition operation (swapping of adjacent characters). The transposition operation has a limitation that cost of transposition be at least the cost of a character deletion plus the cost of a character insertion. This added advantage of finding transposition errors is not of much use for errors in OCR processed text as transpositions are a usual case of typographical errors. For example the Damerau-Levenshtein distance between the strings ‘irec’ and ‘rice’ is 4.

5.3.2 N-Gram Approach

In general, in this approach the words in the dictionary are broken up into sub-sequences of characters of size ‘n’ (‘n’ can be 1, 2, 3...), then a table of is compiled that stores the statistics of the n-grams obtained from the dictionary. When a word has to be checked for spelling, its n-grams are queried for in the n-gram table. If not found, it is then marked as a non-word and spelling correction is applied. For spelling correction step

a similarity criteria is established using the difference in the number of n-grams of a dictionary word and the misspelled word. [39] Describes an N-gram based approach for spell checking.

The Edit Distance approach is expensive because it computes edit distance for each misspelled by comparing with all the entries in target dictionary. N-gram approach on the other hand is comparatively less expensive and this approach is also proven to be useful for errors in the output of OCR process [37].

In the following sections we discuss about two spelling correction approaches that we are going to use in our application. We will first discuss in detail about a new Skip Trie algorithm we developed based on Trie data structure followed by a discussion on Apache's Lucene Framework.

5.4 Skip Trie Matching Algorithm for correcting OCR Extracted Text

Tries are specialized tree data structures where words from a dictionary are stored as a sequence of characters in tree nodes. It was first proposed in 1960 [41]. This data structure has been extensively used for implementing autocomplete feature. Trie data structure is generally used for prefix matching. Reading the word from a Trie involves traversing down the branches of the tree. At each node, the possible completions of the partial word can be found by traversing down all possible paths to the leaf level. The Trie data structure is popular because of the efficient worst-case dictionary lookup times. In a Trie data structure when used for prefix matching, the absolute worst case running time would be $O(n)$, where n is the length of the input string. We have two important reasons for considering a data structure based on Trie for our use case; first, the algorithm should be efficient; second, the limited vocabulary in our dictionary. Since most of the NFT's

are similar in structure and the content they have, the size of our dictionary is at most a few hundred words.

The STM Algorithm described here assumes that a misspelled word which cannot be found in a dictionary has at most ‘n’ character errors, which might be a replacement. The misspelled input word is compared to the dictionary which is stored in memory in the form of a Trie data structure. We call the parameter that identifies ‘n’ as the skip distance and it is a metric we use to find the possible match up of a misspelled input word against words in dictionary. This algorithm is based on the simple idea that Trie data structures efficient storage of dictionary can be used to match a non-word to a dictionary word. Our STM Algorithm has two main operations, one for building the Trie data structure the addString() method and the other for getting suggestions to a misspelled word called skipTrie(). The following Algorithm describes the addString() method which builds a Skip Trie data structure.

```
Procedure: addString(inputStr, curNode)
Params:
        inputStr  : represents the word to be inserted into the Trie data
structure
        curNode  : Pointer to the structure of TrieNode of Trie data structure
    1.  IF length (inputStr) > 0
    2.  c := inputStr[0];
    3.        IF  any(curNode.childNode.char == c)
    4.            curNode = childNode;
    5.            Add(rest(inputStr) ,curNode);
    6.        ELSE
    7.            newNode := new TrieNode();
    8.            newNode.character  := c;
    9.            IF len(inputStr) == 1
    10.                   newNode.wordEnd := True;
    11.           curNode.addChild(newNode);
    12.            Add(rest(inputStr) , newNode);
    13. ELSE return;
```

Figure 15. Algorithmfor addString() method of Skip Trie

A partial in memory representation of the Trie data structure that we can build using the addString() method can be illustrated by the following Figure 16:

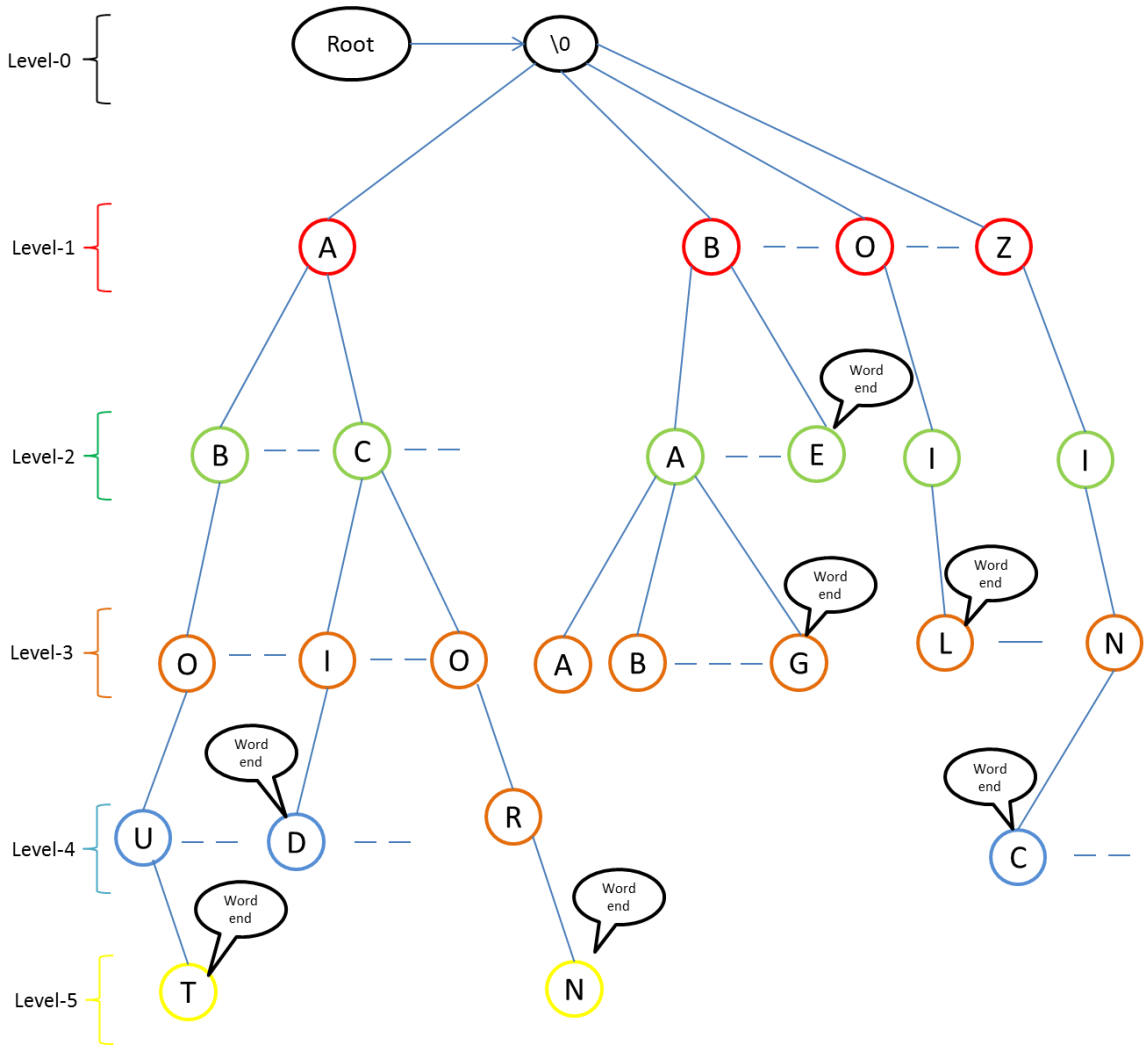


Figure 16. In Memory representation of Trie data structure

The pseudo code of the skipTrie() method is as follows:

```
Procedure skipTrie(inputStr, d, curNode)
Params:
        inputStr  : The misspelled word/ word to get suggestions for
        skipDist  : The skip distance
        curNode   : A node of Trie data structure. (Each node has a character, list of
                    children Trie Nodes and a Boolean wordEnd flag
                    indicating end of word)
```

```
1.  wordToMatch = copy(intpuStr);
2.  suggestionList = [ ];
3.  skipTrie (inputStr, skipDist ,  curNode, suggestion):
4.     IF len(inputStr) == 0 || curNode == NULL: fail
5.     IF len(inputStr) == 1:
6.     IF inputStr[0] == curNode.char || d > 0:
7.         add curNode.char to suggestion:
8.       IF len(suggestion) == len(wordToMatch) &&
9.           curNode.wordEnd == True:
10.          add suggestion to suggestionList;
11.   ELSE IF len(inputStr) > 1
12.     IF inputStr[0] == curNode.char
13.       add curNode.char to suggestion;
14.       nextNode = binSearch(inputStr[1], curNode.chidren)
15.       IF nextNode != NULL:
16.               STM(rest(inputStr), skipDist , nextNode, suggestion);
17.       ELSE IF skipDist > 0:
18.          FOR each node C in curNode.children:
19.             add C.char to suggestion;
20.             STM(rest(inputStr), skipDist-1, C, suggestion);
21.       ELSE fail;
```

Figure 17. Algorithm For skipTrie() method

5.4.1 SkipTrie Asymptotic Analysis

Let len(**inputStr**) = *n* and **skipDistance** = *d*. The largest branching factor of a trie node is $|\Sigma|$, i.e., the size of the alphabet over which the trie is built. If **inputStr** is in the trie, the binary search on line 14 runs exactly once for each of the n characters, which

gives us $O(n \log|\Sigma|)$. If **inputStr** is not in the trie, it is allowed to contain at most d character mismatches. Thus, there are $(n-d)$ matches and d mismatches. All matches run in $((n-d)\log|\Sigma|)$. In the worst case, for each mismatch, lines 18-19 ensure that $|\Sigma|^d$ nodes are inspected, which gives us the run time of $O\left((n-d)|\Sigma|^d \log|\Sigma|\right) = O\left(n|\Sigma|^d \log|\Sigma|\right)$. The worst case rarely occurs in practice because in a trie built for a natural language most nodes have branching factors equal to a small fraction of $|\Sigma|$.

5.5 Skip Trie Example Illustration

To better understand the working of skipTrie Algorithm let us go through a step by step execution of skipTrie algorithm with the help of an example. Suppose that a Trie is populated with the words in dictionary, a partial in memory representation of such as structure can be seen in the Figure 16. The dictionary contains words such as ‘ABOUT’, ‘ACID’, ‘ACORN’, ‘BAG’, ‘BE’,’OIL’, ‘ZINC’ etc. Bubbles at some of the nodes in the figure are used to illustrate the representation of an end of word flag.

Suppose that skip distance is set to 1 and the OCR engine misrecognized the word ‘ACID’ as ‘ACIR.’ The STM starts at the root node, as shown in Figure 18. For each child of the root, the algorithm checks if the first character of the input string matches with any of the root’s children. If no match is found and the skip distance > 0, the skip distance is decremented by 1 and the recursive calls are made for each of the root’s children. If there is a match, as in this case when ‘A’ in the input matches the root’s ‘A’ child. Since the match is successful, a recursive call is made on the remainder of the input ‘CIR’ and the root node’s ‘A’ child at Level 1 as the current root node, as shown in Figure 19:

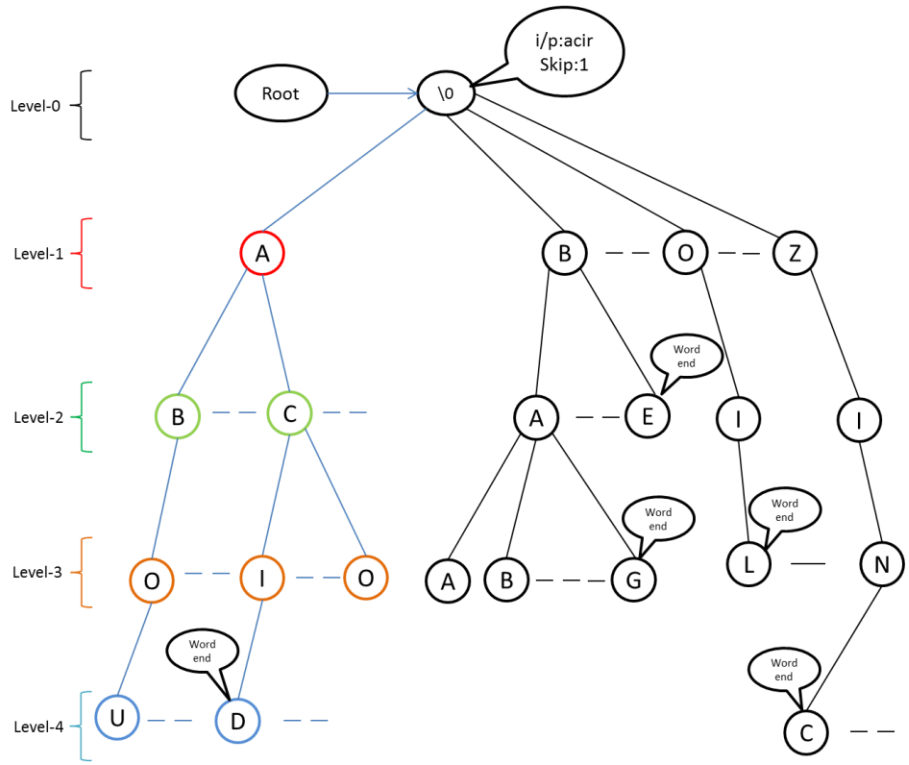


Figure 18. Illustration of Skip Trie Execution – 0

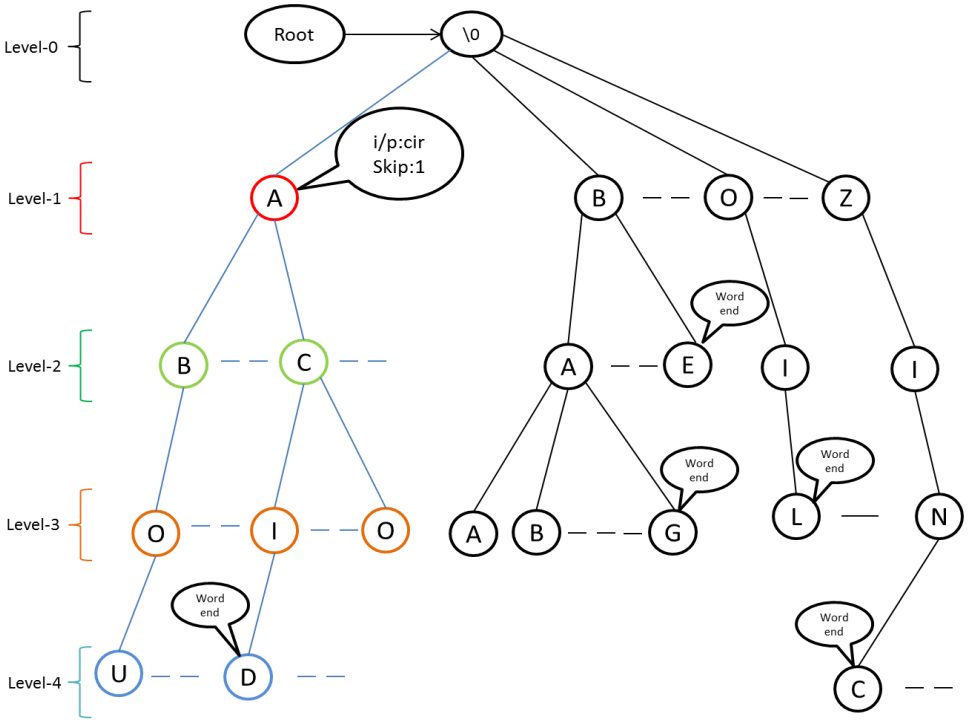


Figure 19. Illustration of Skip Trie Execution - 1

The algorithm matches the current character ‘C,’ i.e., the first character of the truncated input ‘CIR’ with the first child of the current root ‘A’ at Level 1 which is ‘B’. Since the match is unsuccessful and because skipdistance is 1, the algorithm decrements the skipdistance by 1 making it a 0 and then continues the execution with truncated input word ‘IR’, with the current root as node ‘B’. This is represented in Figure 20.

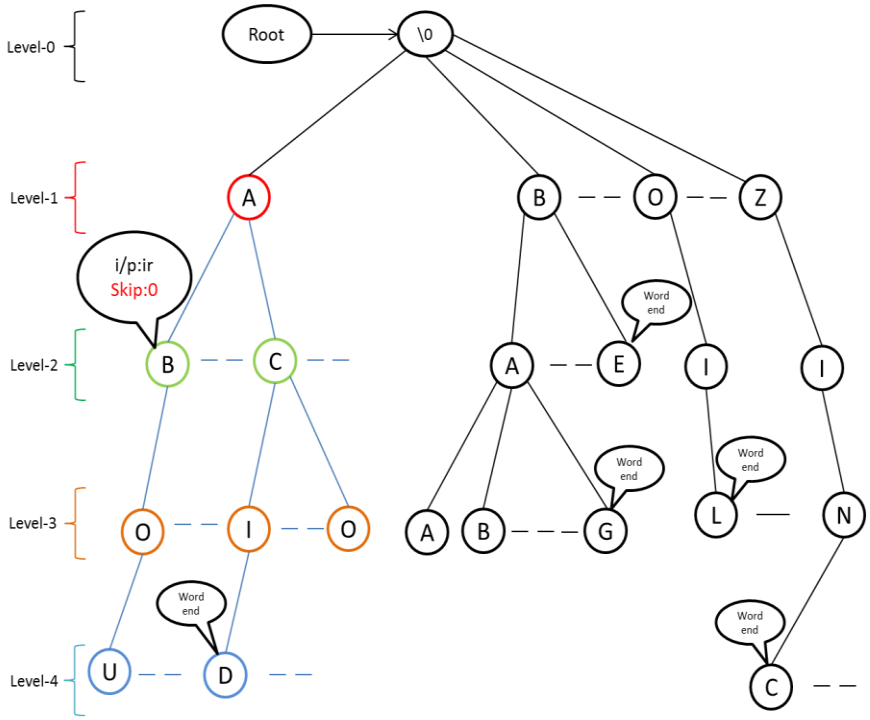


Figure 20. Illustration of Skip Trie Execution – 2

Now, the algorithm tries to match the first character ‘I’ of the truncated input word ‘IR’ with all the children of the root ‘B’. Since ‘B’ does not have a child node ‘I’, and because the skipdistance is now 0 the algorithm backtracks to Level 1 with the

current root now being ‘A’ and truncated input string “CIR”.Also the skipdistance will be 1. This is represented using the Figure 21.

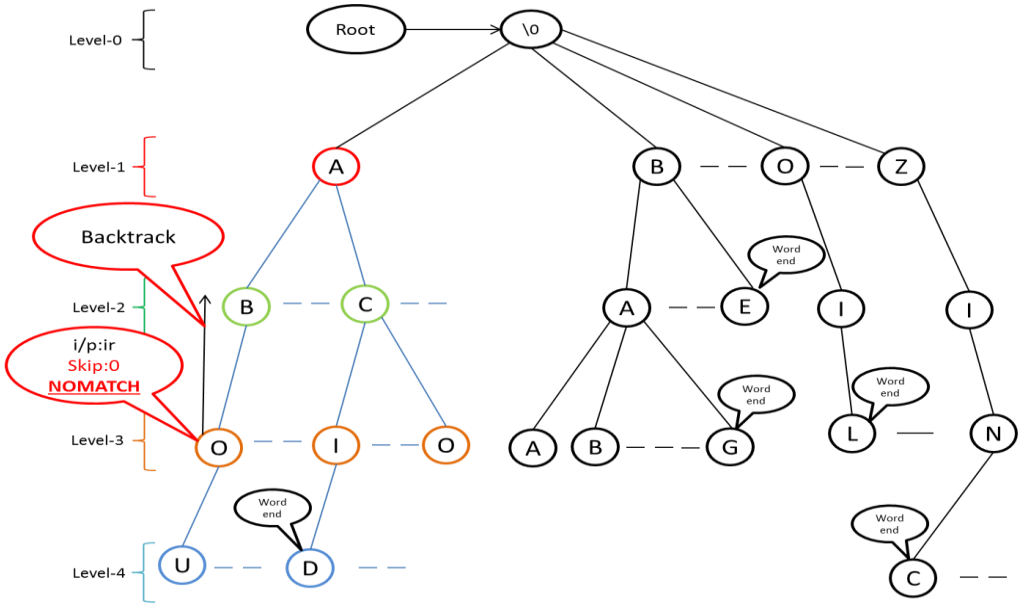


Figure 21. Illustration of Skip Trie Execution – 3

The algorithm matches the current character ‘C,’ i.e., the first character of the truncated input ‘CIR’ with the right child of the current root ‘A’ at Level 1, truncates the input to ‘IR’ after the match, and recurses to the node ‘C’ at Level 2, i.e., the right child of the node ‘A’ at Level 1. The skip distance is still 1, because no mismatched characters have been skipped so far. This is represented using Figure 22.

The matching of the input strings current character ‘I’ with the left child of the node ‘C’ at Level 2 results in the truncation of the input to ‘R’ and a recursive call to the node ‘I’ at Level 3 and the skip distance still 1, as shown in Figure 23.

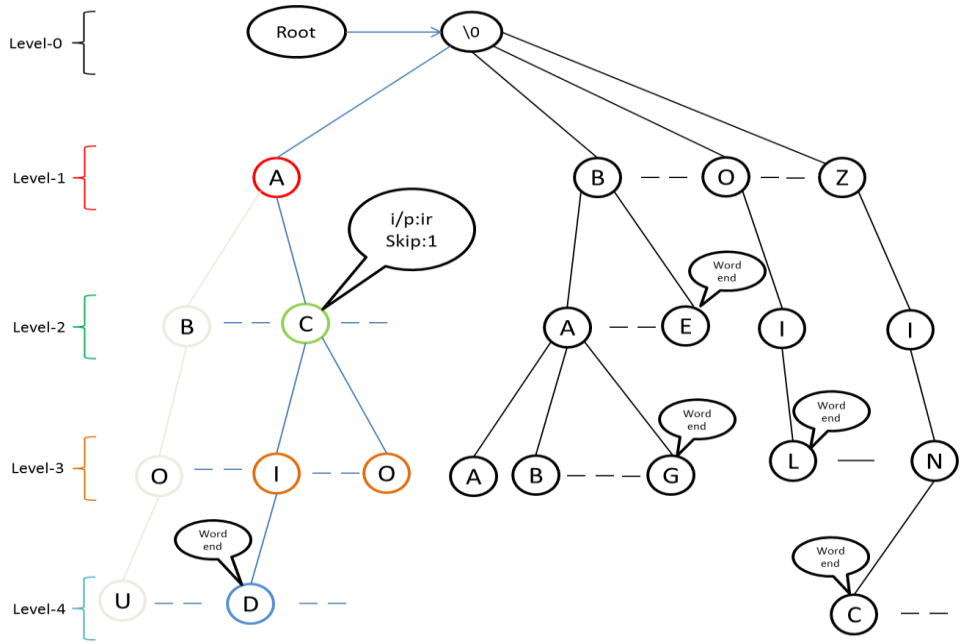


Figure 22. Illustration of Skip Trie Execution - 4

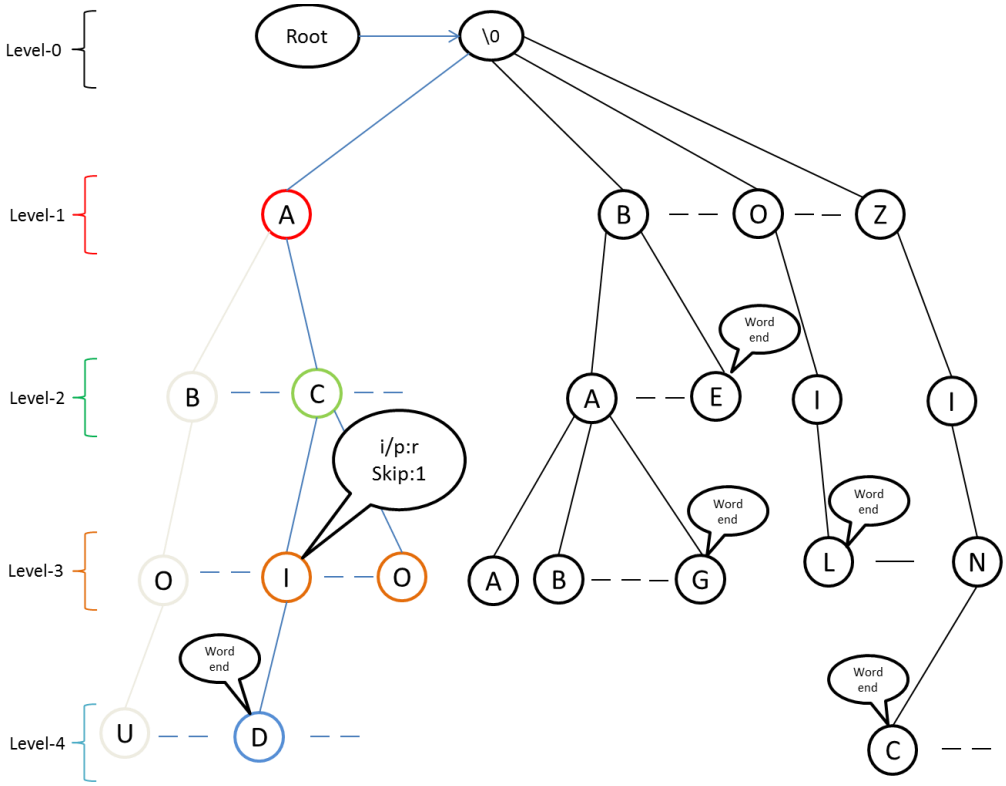


Figure 23. Illustration of Skip Trie Execution - 5

The last character of the input string, ‘R,’ is next matched with the children of the node ‘I’ at Level 4 (See Figure 6). The binary search on the node’s children fails. However, since the skip distance is 1, i.e., one more character can be skipped and the skip distance is decremented by 1. Since there are no more characters in the input string, the algorithm checks if the current node has a word end flag set to true. In this case, it is, and the matched word, ‘ACID,’ is added to the returned list of suggestions. Figure 24 shows this.

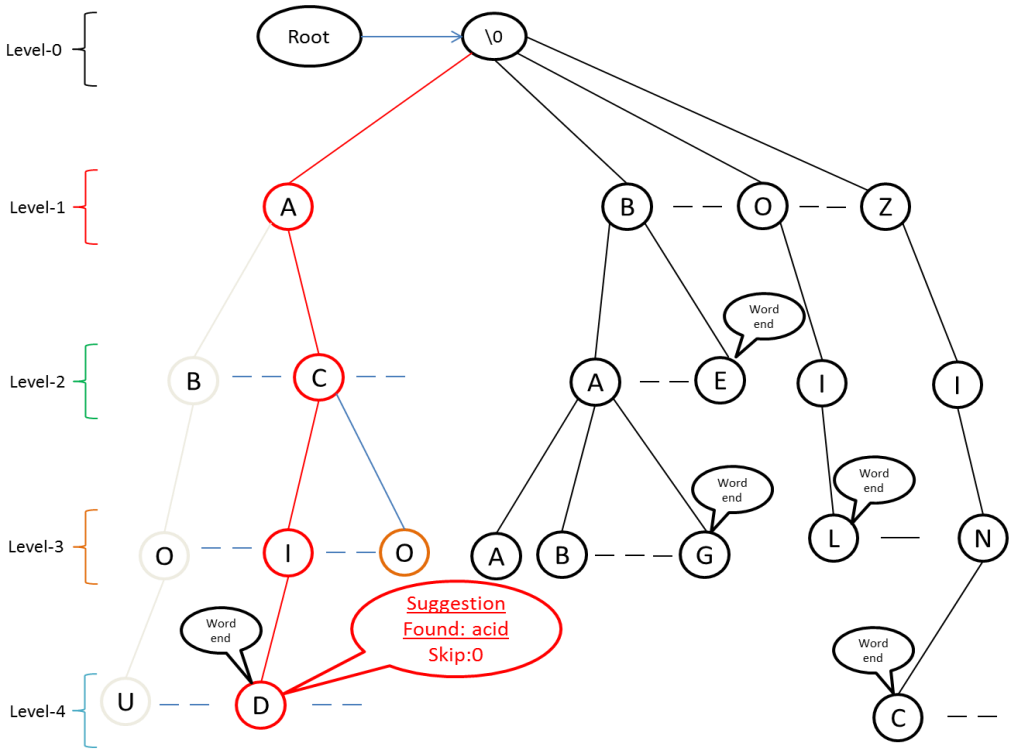


Figure 24. Illustration of Skip Trie Execution – 6

Let us consider a case where the error in the misspelled word is not the last character. For example the input string is ‘ACBD’, it has an error in the 3rd character

(when compared to dictionary word ‘ACID’). If the skip distance is 1, during step 3, the algorithm tries to check if any of the children of node ‘C’ match with ‘B’ and fails to find a match. As the skipdistance is 1 the algorithm checks each node that is child of node ‘C’(level -2) chooses to skip matching the node at level-3 and tries to find a possible match for a node with the next character ‘D’ in level-4 with the word end flag set. Since it will be found a suggestion is added to the returned list of suggestions.

The current implementation of our Skip Trie has one limitation, it can only find suggestions for misspelled words that have replacement errors, for example, if the misspelled word is ‘ietary’ instead of ‘dietary’ the Skip Trie will find the suggestion as ‘dietary’ incase the skip distance is chosen as 1. If the misspelled word is ‘etary’, ‘dietary’ will not be given out as a suggestion whatever the skip distance might be i.e., the length of the misspelled word should be the same as that of similar dictionary word.

This Skip Trie algorithm has been implemented in our Android client application, the details of which can be found in the Chapter 6 that follows next.

5.6 Apache Lucene

Let us now look at the other Spelling correction framework that we use in our Android application. Apache’s Lucene is an open source project [33]. Primarily Lucene is a full text search engine API. Lucene project included a Spell Checker API to provide functionality similar to famed Google’s “Did you mean” for Apache’s SOLR project. Lucene’s Spell Checker API is based on N-Gram approach. Lucene builds an index of N-grams of the target dictionary. It then uses this index to calculate the similarity between a misspelled word and words in dictionary to fetch the most relevant words. Out of these relevant words, Lucene uses one of the three available algorithms to calculate a distance

metric (how similar the specified strings are to one another) which it then uses to find a bunch of suggestions. The three algorithms that are used in Lucene are Levenshtein distance [36], N-Gram distance [40] and JaroWinkler distance. In our application we use Lucene's spell correction capability to process the output of Tesseract on Android device using Lucene's Levenshtein distance algorithm and N-Gram distance algorithm. More details on how Lucene is used in our application can be found in the Chapter 6 that follows next.

5.6.1 Time Complexity analysis of Algorithms in Lucene

Lucene first uses N-Gram matching to find relevant words of a given misspelled word. Let us suppose that it generates M suggestions for string S1. Lucene uses a dynamic programming algorithm to compute Levenshtein distance, which has a time-complexity as $O(|S1| * |S2|)$ where |S1| and |S2| are the strings being matched, i.e. $O(N^2)$ if S1 and S2 are of same length 'N.' So the total time complexity of Lucene's spellchecker using Levenshtein distance as a distance metric is $O(M * N^2)$. The space-complexity is also $O(|S1| * |S2|)$ if the whole of the matrix is kept for a trace-back to find an optimal alignment. Since in the case of Levenshtein distance in Lucene we only consider the edit distance, only two rows of the matrix need be allocated and they can be reused, so the space complexity is then$O(|S1|)$, i.e. $O(N)$.

Lucene's N-Gram distance algorithm slices the misspelled word into chunks of size 'n' (default 2) and compares them to an index of a dictionary of grams [40]. So the time complexity of using N-gram approach will be O (no of n-grams in the dictionary* number of grams in misspelled word).

CHAPTER 6

APPLICATION DESIGN AND IMPLEMENTATION

6.1 Overview

An Android application has been developed to understand the implications of different binarization algorithms and for comparing the performance of Skip Trie algorithm to Lucene spelling correction algorithms. This application has four key modules.

1) Binarization Module
2) Tesseract OCR Engine
3) Spelling Correction Module
4) PHP module

The first three modules run on the device itself. There is a provision for the user to choose between running Tesseract either on the device or on a server. If the user chooses to upload images to a server for processing, only the OCR part (Tesseract) is executed on the server. The PHP Module is used to perform the OCR on the server and send back the recognized text to the Android client; the rest of the processing still takes place on the Android device. The application also has a provision for the user to choose between the Skip Trie Algorithm and the Apache Lucene for a spell correction engine.

6.2 System Design

A high level overview of the flow of control between the different modules in the Android client application is shown in the Figure 25. Figure 26 shows the high level structure of the application classes it also shows the relationships and attributes of different classes used in our application. Although for each image the processing happens

as a series of sequential steps, to optimize the performance we set each image as a separate thread.

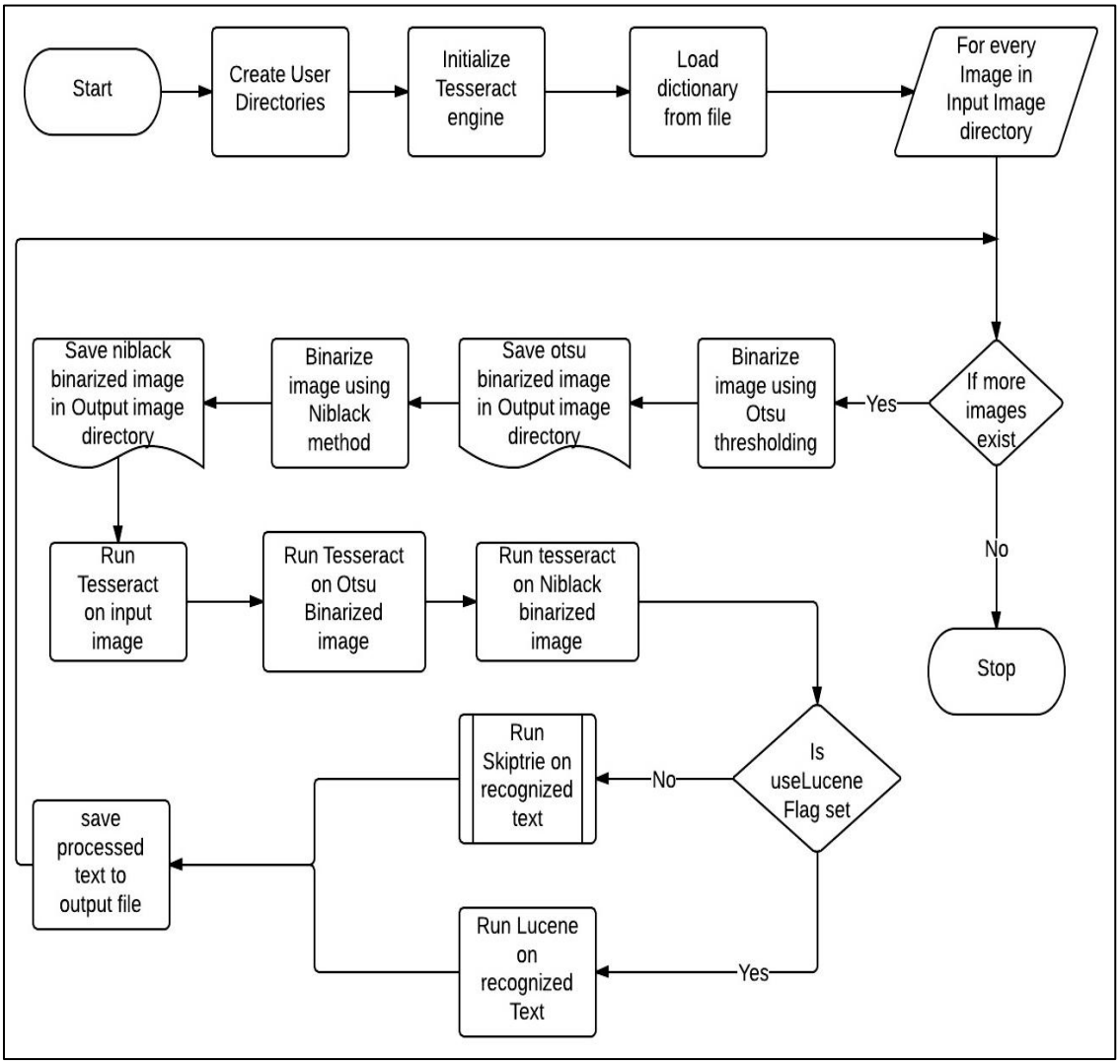


Figure 25. Flow Chart of Android Client Application

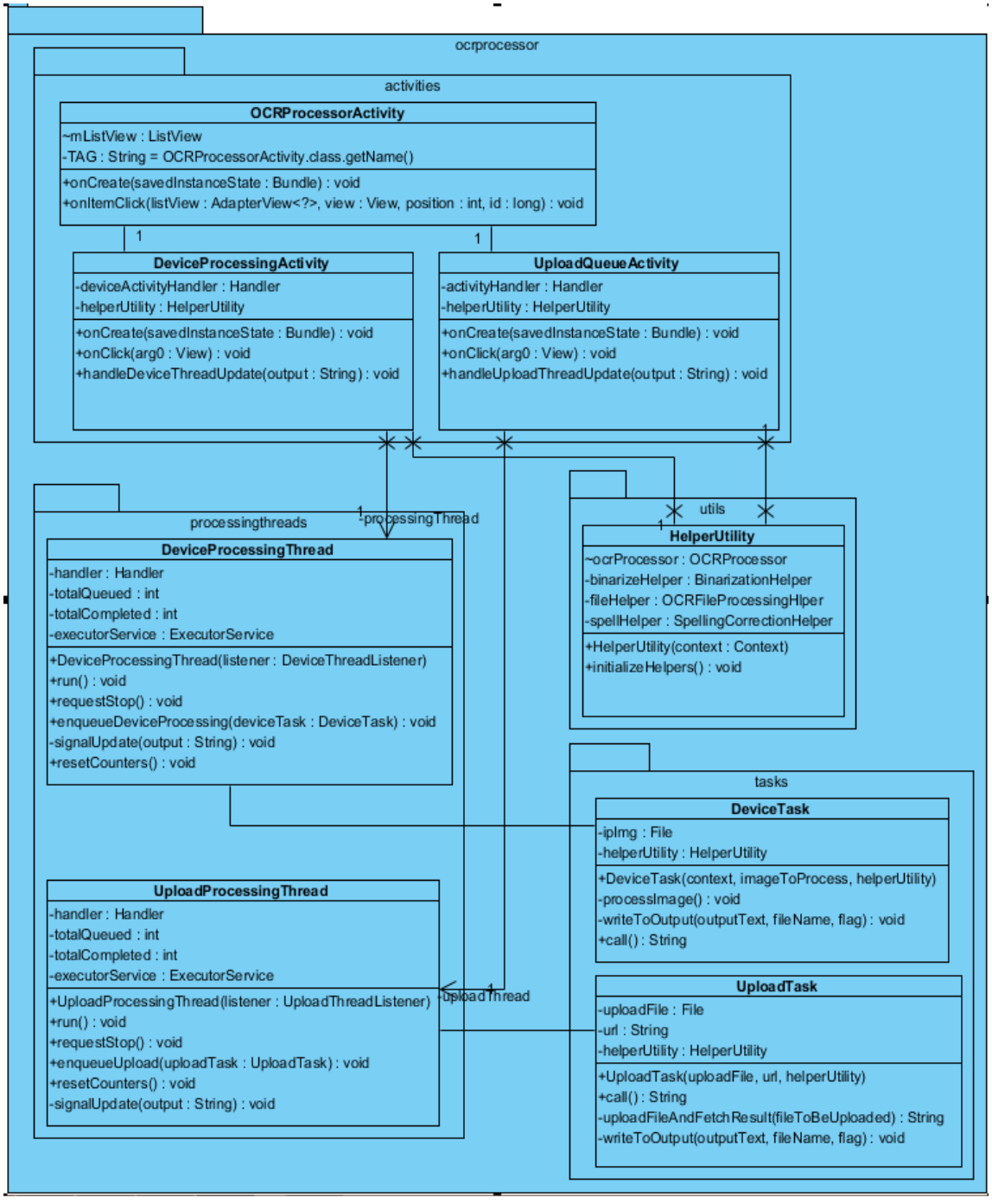


Figure 26. Class diagram of Android Client Application

The following set of sequence diagrams clearly detail the processing involved during processing the file on both device and on server.

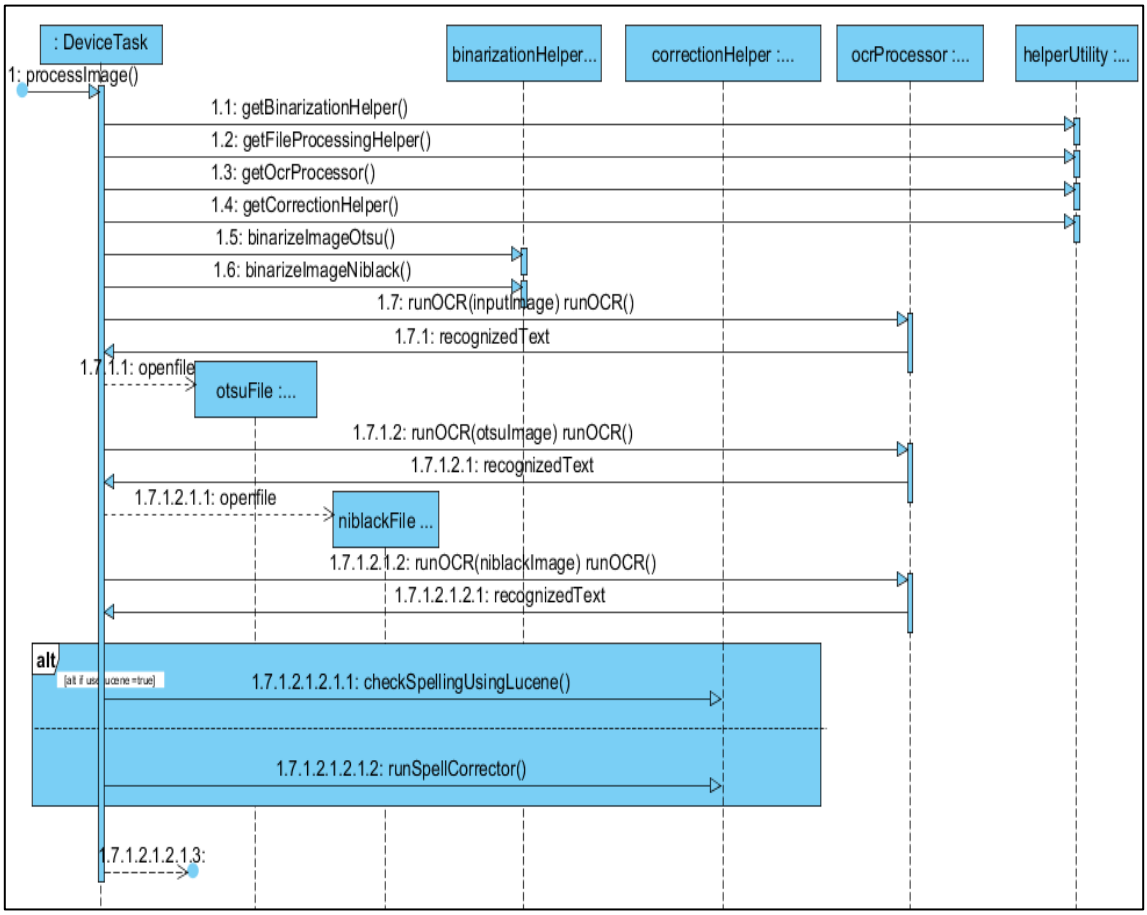


Figure 27. Sequence of Interactions of processImage() method of Device Task

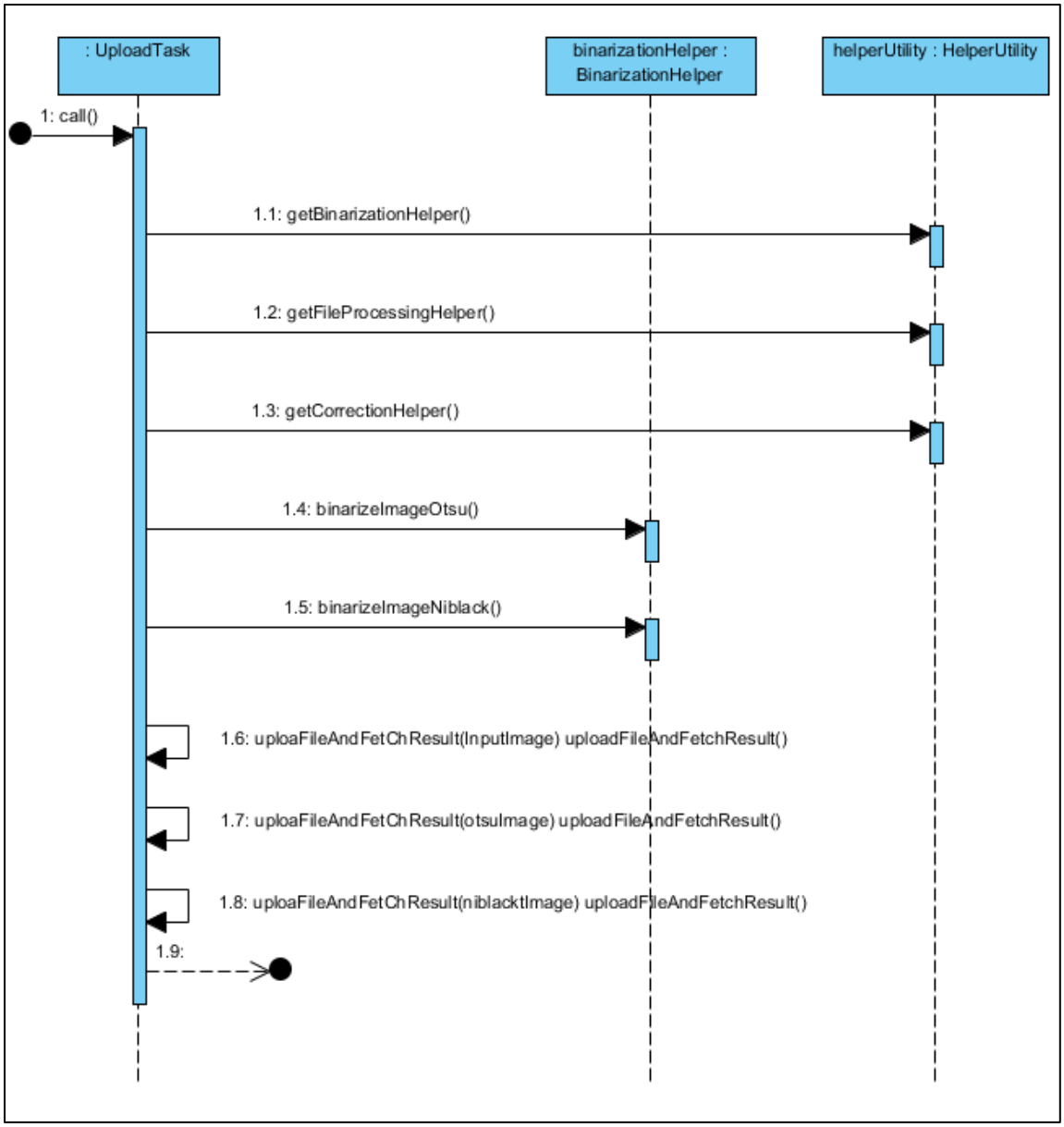


Figure 28. Sequence of Interactions of the Call method of Uploadtask

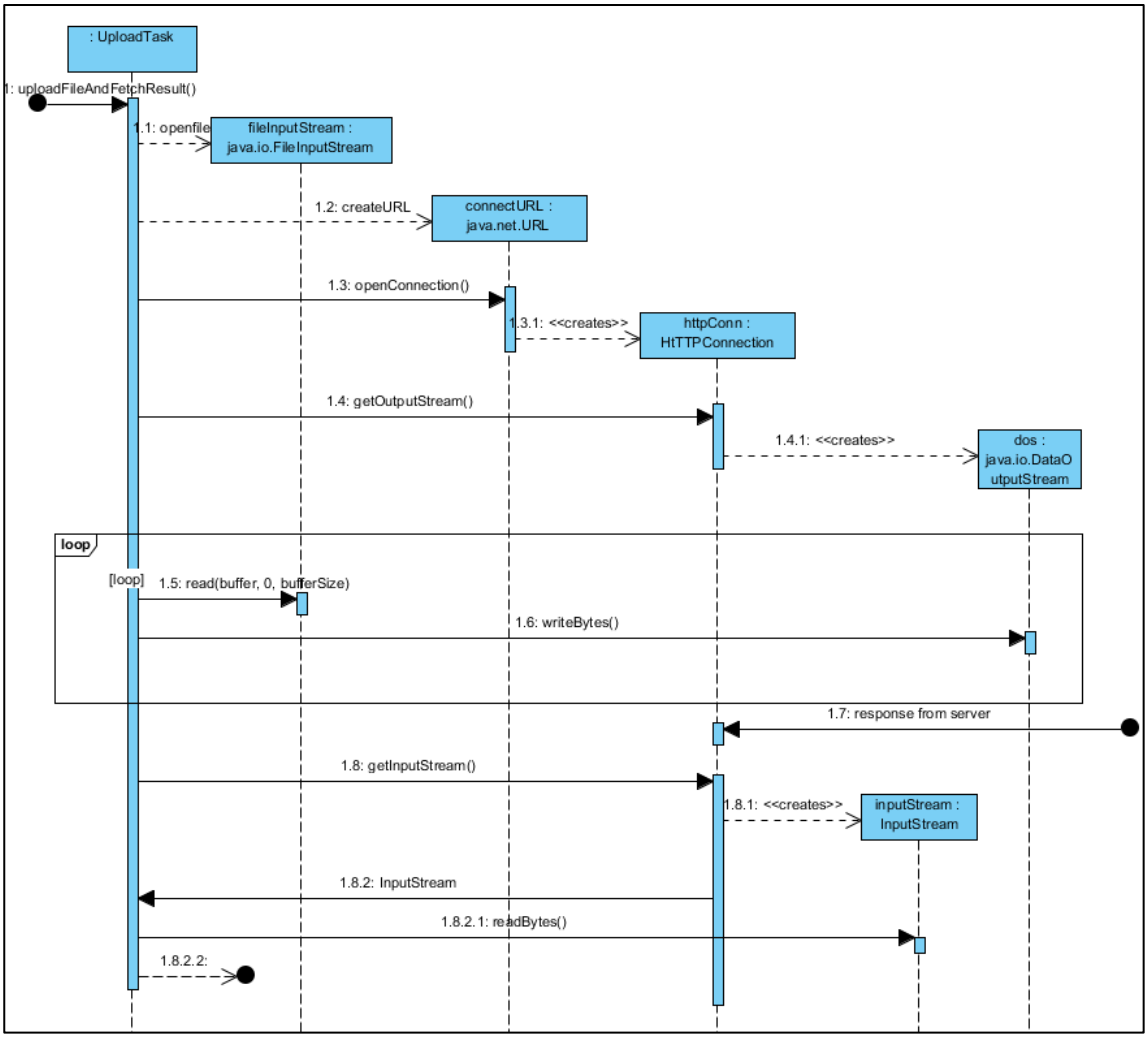


Figure 29. Sequence of Interactions of uploadFileAndFetchResult of UploadTask

6.3 Implementation

Once the app starts it follows the following sequence of steps: First the app creates the necessary temporary directories and output files; it also ensures the presence of necessary training files for Tesseract on the SD Card. Both the modified Otsu's method and modified Niblack's method of binarization are applied to each image that is read from the input image directory on SD Card; these binarized images are stored in the binary images directory on the SD Card. Next, based on the user choice from the main

menu, all three images (input image and the two binarized images) are subjected to the OCR process using Tesseract either on the device itself or on the server. The recognized text is obtained and is subjected to spelling correction again based on a flag set by the user. Finally, the processed text is both saved to a text file and displayed to the user. To optimize the performance, the process mentioned above is entirely processed as a separate thread for each image.

6.3.1 Tesseract on Server

We use HTTP protocol to send the images from the device to server. We are using Ubuntu 12.04 Linux system for a server. The server is running Apache webserver [43] with required PHP packages installed. Tesseract, and its supporting packages such as Leptonica and Imagemagick [44], are also installed on the server. When an image from the Android client application is sent to the server, a PHP script captures the image and triggers OCR on the image. It then captures the extracted text and sends it back to the Android client application where rest of the processing continues.

6.3.2 Handling Long Running operations

Android applications have problem with long-running operations on the UI Thread. The application force closes if an operation takes more than 5 seconds while running on the UI Thread. Since in our application we have some long-running operations such as OCR processing and network communication, we have decided to use the Pipeline Thread model. The Pipeline Thread holds a queue of units of work that can be executed. Other threads can push new tasks into the Pipeline Thread’s queue when new tasks are available. The Pipeline Thread processes the tasks in the queue one after another. If there are no tasks in the queue, it blocks until a new task is added to the queue.

In Android this design pattern can be easily implemented using the Loopers and Handlers. A Looper class makes a thread into a pipeline thread and the Handler makes it easier to push tasks into a Looper from other threads. We have DeviceTask and UploadTask classes that take care of entire processing right from binarization to OCR to applying spell correction. Then, the DeviceProcessingThread and UploadProcessingThreads both have an interface that allows to enqueue DeviceTask instances and UploadTask instances respectively. There are also the DeviceThreadListener and UploadThreadListener interfaces that allow the DeviceProcessingThread and UploadProcessingThreads to notify DeviceProcessingActivity and UploadQueueActivity about the status updates respectively. In our application, the DeviceThreadListener and UploadThreadListener interfaces will be implemented by the DeviceProcessingActivity and UploadQueueActivity because we want to reflect the progress of processing in the UI.

In our application, to get better accuracy from the Tesseract OCR engine, we have applied a few tweaks while initializing the Tesseract API. Tesseract's API provides an option to whitelist or blacklist certain characters. Whitelisting characters will limit the set of characters that Tesseract is looking for. Blacklisting on the other had will instruct Tesseract not to look for certain characters in the image. By manually analyzing several NFT's we have prepared a list of characters that occurred in the NFT's and used them for whitelisting.

6.4 Application Demo

As soon as the application launches, the following landing screen is displayed. The main screen of the application is shown in the Figure 30 below.

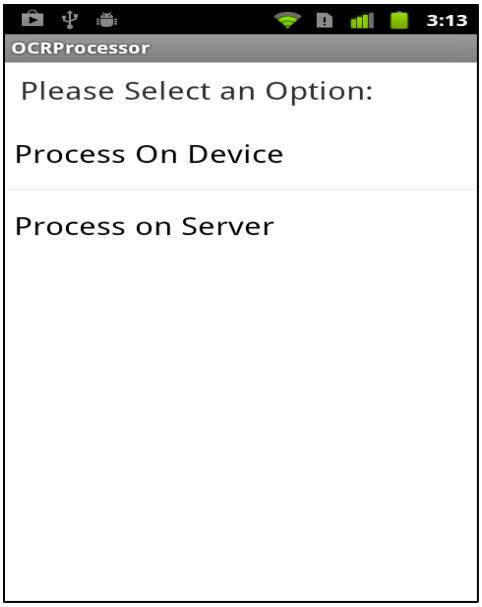


Figure 30. Application Demo Screenshot – 1

Choosing ‘Process on Device’ tab takes the user to the screen shown in Figure 31.

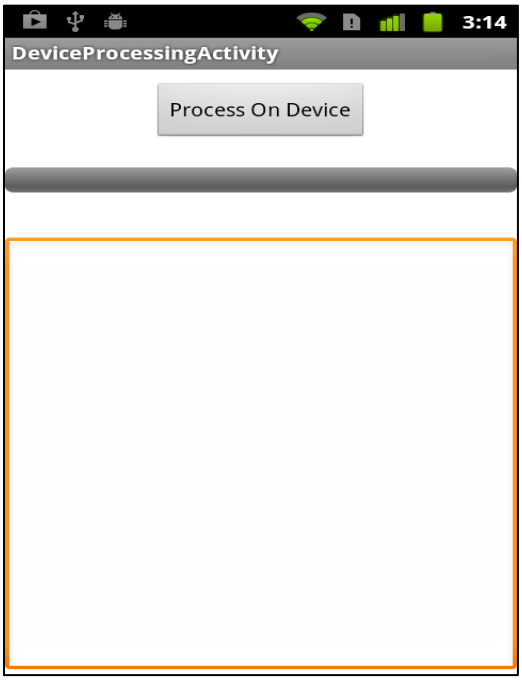


Figure 31. Application Demo Screenshot – 2

When the user clicks on the ‘Process on Device’ button the application starts processing the images gradually showing the progress on the progress bar as shown in the Figure 32 below.

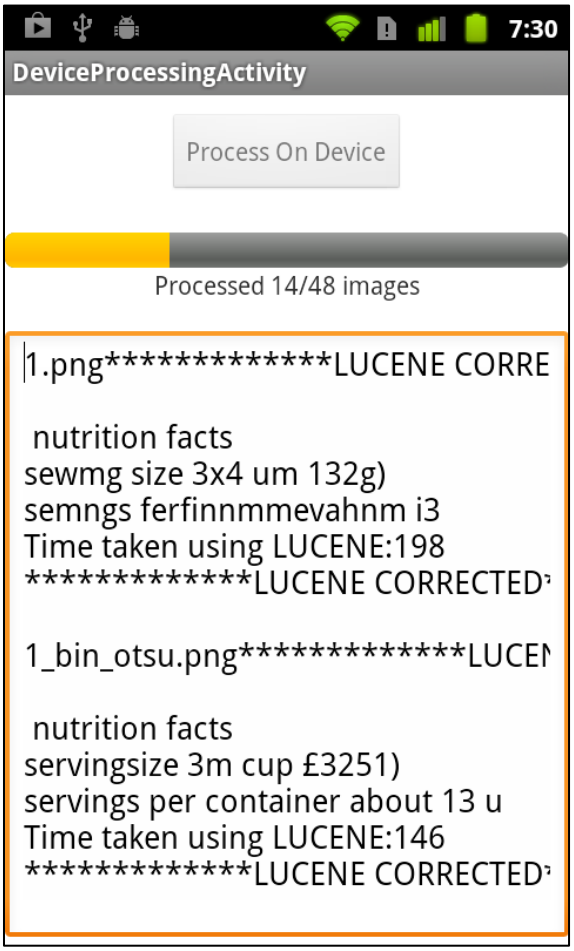


Figure 32. Application Demo Screenshot - 3

If the user selects the ‘Process on Server’ tab on the landing screen, the application will show the following screen, which is identical to the device processing tab option. The Upload process screen looks as shown in the Figure 33 below.

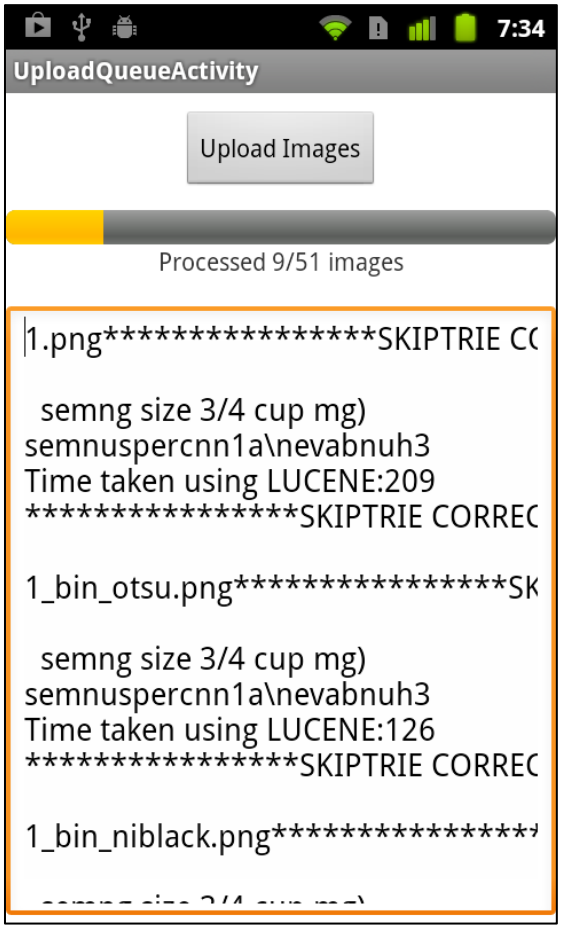


Figure 33. Application Demo Screenshot - 4

CHAPTER 7

RESULTS ANALYSIS

In this Chapter we try to study the findings by running our experiments on a sample of line images. We have evaluated more than 200 line images obtained from segmenting more than 30 NFTs using the technique discussed in Section 4.2.3. These NFTs were captured from regular items available in a grocery store. The reminder of this Chapter is organized as follows: first we give insights into the effect of Otsu's and Niblack's binarization on the output of Tesseract. We then compare the performance of the Skip Trie algorithm with the Lucene spell corrector and discuss our findings.

7.1 Impact of Otsu's Method and Niblack's Method on OCR Output

Running time performance

The one major constraint that we have set in our application is to minimize the time taken during pre-processing. We logged the running times of both Otsu's method and Niblack's method of binarization in our experiments and found that the Niblack's method is slower when compared to that of the Otsu's method. The average time taken to binarize the images in our sample using Niblack's method is 9 milliseconds, whereas the average time taken by Otsu's method is 1.9 milliseconds. One apparent reason why Otsu's method is faster is because it is implemented at the native (C code) layer, whereas Niblack's method is implemented in java. Another observation is that the runtime difference between the two methods significantly increases as the image resolution increases.

OCR Performance

All the images in our sample are subjected to both Niblack's and Otsu's method of binarization and then passed onto the Tesseract OCR to extract the text from them. Out of 200 line images of our sample, the Tesseract could not extract any text from 20 images that were binarized using Niblack's method. Tesseract could extract at least some text, even though it may be garbled, from the same images binarized using Otsu's method. So in our experiment 10% of images binarized using Niblack's method completely failed at OCR. An example of two such images is shown in Figure 34.

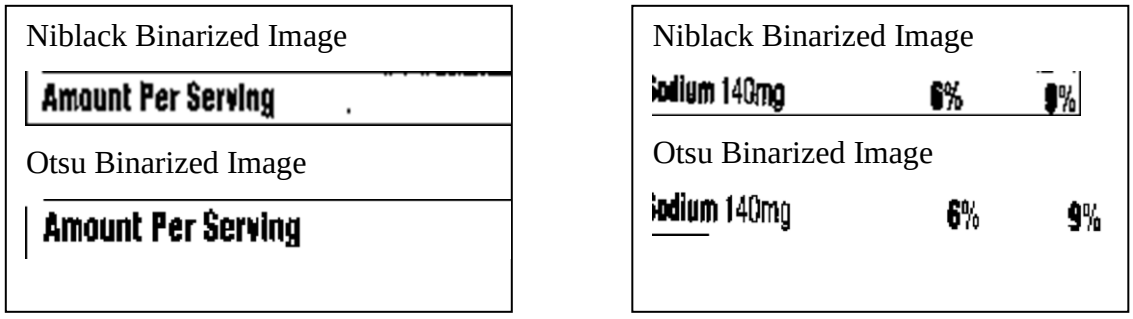


Figure 34. Comparision of Binarized images

7.2 Skip Trie vs. Lucene Comparison

We ran multiple tests to compare the performance of the Skip Trie algorithm with two implementations of established spelling correction algorithms in the Apache Lucene Framework using the Levenshtein Distance and N-Gram distance. Both running time and accuracy are considered as metrics for performance evaluation. Our sample size is 600 line images (both binarized and non-binarized) for this experiment. Tesseract extracts text

from these 600 images and the extracted text is then passed to the Skip Trie algorithm as well as to the Apache Lucene.

Running Time Comparison

Minimizing the time taken during the post-processing stage is very important if we want to process multiple images simultaneously and make this application useful. To minimize the running time we have implemented the Skip Trie algorithm, which is efficient in searching limited vocabulary dictionary.

The Average time taken by the Skip Trie algorithm is 24.66 milliseconds. Average time taken by Lucene when using Levenshtein distance is 50.47 milliseconds, whereas the Average time taken by Lucene when using N-Gram distance is 50.79 milliseconds. From our experiments we conclude that our skip time algorithm runs at least 50% faster than the Lucene's spell checker.

Accuracy Comparison

Although running time is an important metric of performance, for a spell checker being able to correct more number of misspelled words is a much more important metric. For the purpose of comparison of the ability to correct non-word errors we use the Recall parameter. We define recall as follows:

$$Recall = \frac{(number\ of\ corrected\ words)}{(number\ of\ misspelt\ words)}$$

From our experiments we calculated the Recall of the Skip Trie algorithm as 0.15. The Recall of Lucene when using the Levenshtein distance is 0.078, and the Recall when

using the N-Gram distance is 0.085. In other words, about 15% of the non-word errors were corrected when using the Skip Trie algorithm, whereas a little above 7% and 8% of non-word errors were corrected by Lucene when using the Levenshtein distance and N-Gram distance, respectively.

It is clear from the above results that the Skip Trie algorithm performs better than Lucene for correcting non-word errors in our case. Also, Skip Trie's performance in terms of running time is better than that of Lucene. One important observation is that even though Skip Trie only works for misspelled words that are of the same length as that of similar dictionary words, it performed better than Lucene.

CHAPTER 8

FUTURE WORK

This chapter discusses future improvements to the STM algorithm, as well as useful extensions to this project for the future.

8.1 Improvements to Skip Trie Algorithm

The STM algorithm in its current form generates suggestions from the target dictionary that have the same length as that of the misspelled word. Even though this algorithm produces suggestions for misspelled words based on the input skipdistance parameter, it only controls the number of misrecognized characters from the misspelled word when compared to a target dictionary word. This algorithm can be improved to suggest words that are not only of the same length as the misspelled word, but also other similar words from the target dictionary that are of different lengths. Using this enhancement, the STM algorithm can be used to detect non word errors such as those that occur from character splitting during the OCR process; one such example is “potassium” recognized as “potassillm.” This enhancement will increase the recall rates of the STM algorithm, as character splitting is a common problem in OCR output.

The STM Algorithm can also be improved to decode joined words and words with punctuation symbols in the OCR output. For example, if OCR recognizes “Nutrition Facts” as ”NutritionFacts” or “Nutrition,Facts,” the algorithm should be able to decode the individual words “Nutrition” and “Facts” separately. The STM algorithm is not context aware in the sense that this algorithm only checks for the correctness of spelling based on a dictionary and not the correctness of a word in a context. For example this algorithm does not try to correct the word “Fats” in the extracted text “Nutrition Fats” as

it is a dictionary word; in an ideal case the algorithm should produce “Facts” as a possible suggestion. The STM algorithm can be improved to produce context sensitive suggestions.

8.2 Nutrition Information Management

A database management module can be added to this application that uniquely stores a product along with all the processed nutritional and allergen information. This will alleviate the need to go through the entire OCR Process for one processed product. This enhancement will also create a scope for automatic creation of large database nutrition management information by pooling in information from all the users of the application.

8.3 Hardware Capability Utilization

Most of the smartphones being produced in the recent past have multi-core processors. The STM Algorithm can be improved take advantage of this multi-core architecture and try to process multiple words of the OCR output in parallel, thereby minimizing the time spent in post-processing the OCR output. This multi-core capability can also be leveraged while pre-processing the image there by enhancing the quality of output produced by Tesseract.

CHAPTER 9

CONCLUSION

In this report we first discuss about the importance of extraction of nutritional information and allergen information for the VI grocery shopper. Chapter 2 presented an overview of the related work done on the assistive technology systems. We discussed the existing systems such as the ShopMobile2, Grozi, and similar systems built at CSATL and various other institutions for use in assistive technology. Chapter 3 presented an evaluation of existing non-commercial OCR solutions available for the Android mobile platform to extract the nutrition information from an image. Our experiments in Chapter 3 suggest that the Tesseract OCR engine performed better of the two solutions considered. In Chapter 4 an overview of different stages involved in an OCR process are discussed. Processing that happens at each stage is discussed, and in particular the advantage of binarization algorithms for preprocessing the image and a detailed explanation of the algorithm used in NFT localization and segmentation were discussed. Chapter 5 presented the analysis of the task at hand and discussed in detail the implementation of the modified Niblack's method and modified Otsu's method of binarization algorithms used in our application. Also, details about the spelling correction algorithms used in our application were provided. In particular, a detailed explanation of our SkipTrie Matching Algorithm, along with a clear analysis of its process, was provided with the help of an example. We also discussed Apache Lucene and the spelling correction used in Chapter 5. Chapter 6 provided the system design and implementation details of our application. Class structure, sequence of interactions between important modules, and specific implementation details were presented in Chapter 6. Chapter 7

presented the findings of our experiments using the implementation described in Chapter 6. In Chapter 7, we compared the effect of different binarization methods used in our application. We also compared the performance of the our STM algorithm to an open source spelling correction module in the Apache Lucene framework, and concluded that the STM algorithm performed better both in terms of time taken to process as well as the number of non-word errors corrected in our application. Chapter 8 presented possible future enhancements to our application. Integrating our OCR module and our spelling correction module into applications such as ShopMobile2 will create a usable and practical nutrition management system for the Visually Impaired shoppers.

REFERENCES

[1] American Academy of Ophthalmology, http://www.aao.org/newsroom/upload/Eye-Health-Statistics-April-2011.pdf, Retrieved April 15, 2013.

[2] A. S. Helal, S.E. Moore, and B. Ramachandran, "Drishti: An integrated navigation system for visually imparired and disabled," in ISWC '01: Proceedings of the 5th IEEE International symposium on Wearable Computers, p. 149. Washington, D.C, USA: IEEE Computer Society,2001.

[3] Kutiyanawala, A., Kulyukin, V., and Nicholson, J. (2011). Toward Real Time Eyes-Free Barcode Scanning on Smartphones in Video Mode . Proceedings of the 2011 Rehabilitation Engineering and Assistive Technology Society of North America Conference (RESNA 2011), Toronto, Canada.

[4] Kulyukin V, Gharpure C, Nicholson J. RoboCart: Toward robotassisted navigation of grocery stores by the visually impaired. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Edmonton, Canada: IEEE Press 2005; pp. 2845-50.

[5] Nicholson J, Kulyukin V. ShopTalk: Independent blind shopping = verbal route directions + barcode scans. Proceedings of the 30th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA). Phoenix, Arizona 2007.

[6] Merler M, Galleguillos C, Belongie S. Recognizing groceries in situ using in vitro training data. SLAM, Minneapolis, MN 2007.

[7] Krishna S, Panchanathan S, Hedgpeth T, Juillard C, Balasubramanian V, Krishnan NC. A wearable wireless rfid system for accessible shopping environments. 3rd Intl Conference on BodyNets'08; Tempe, AZ 2008.

[8]Lanigan PE, Paulos AM, Williams AW, Rossi D, Narasimhan P. Trinetra: Assistive technologies for grocery shopping for the blind. In: International IEEE-BAIS Symposium on Research on Assistive Technologies (RAT). Dayton, OH 2007.

[9] Narasimhan P. Assistive Embedded technologies. IEEE Computer 2006; vol. 39: pp. 85-87.

[10] Kulyukin, V., and Kutiyanawala, A., From ShopTalk to ShopMobile: Vision-Based Barcode Scanning with Mobile Phones for Independent Blind Grocery Shopping. Proceedings of the 2010 Rehabilitation Engineering and Assistive Technology Society of North America Conference (RESNA 2010), Las Vegas, NV.

[11] ShopMobile2, http://www.aliasgar.info/Shopmobile2.html, Retrieved April 15, 2013.

[12] Kutiyanawala, A., Kulyukin, V., Nicholson, J.: Teleassistance in Accessible Shopping for the Blind. In: Proceedings of the 2011 International Conference on Internet Computing, July 18-21, pp. 190–193. ICOMP Press, Las Vegas (2011)

[13] ABBYY Mobile OCR Engine. http://www.abbyy.com/mobileocr/, Retrieved April 15 2013.

[14] Wikipedia, OCR, http://en.wikipedia.org/wiki/Optical_character_recognition, Retrieved April 15, 2013.

[15] WINTONE Mobile OCR Engine. http://www.wintone.com.cn/en/prod/44/detail270.aspx, Retrieved April 15, 2013.

[16] X. P. Luo, J. Li, and L. X. Zhen. Design and implementation of a card reader based on build-in camera. In ICPR ’04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR’04) Volume 1, pages I: 417–420. IEEE Computer Society, 2004.

[17] Gaudissart, V., Ferreira, S., Thillou, C., and Gosselin, B. Sypole: mobile reading assistant for blind people. In Proceedings of European Signal Processing Conference,EUSIPCO (2005).

[18] Hairuman, I., and Foong, O.-M. Ocr signage recognition with skew and slant correction for visually impaired people. In Hybrid Intelligent Systems (HIS), 2011 11th International Conference on (December 2011), pp. 306{310.

[19] Tesseract OCR Engine. http://code.google.com/p/tesseract-ocr/, Retrieved April 15, 2013.

[20] GOCR - A Free Optical Character Recognition Program. http://jocr.sourceforge.net/, Retrieved April 15,2013.

[21] Smith, R., "An Overview of the Tesseract OCR Engine," Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on , vol.2, no., pp.629,633, 23-26 Sept. 2007

[22] Joshi, Anand, Mi Zhang, Ritesh Kadmawala, Karthik Dantu, Sameera Poduri, and Gaurav S. Sukhatme. "OCRdroid: A Framework to Digitize Text Using Mobile Phones." In ICST International Conference on Mobile Computing, Applications, and Services. 2009

[23] ShopMobilePoster, http://digital.cs.usu.edu/~vkulyukin/vkweb/pubs/RESNA2010_ShopMobilePoster.pdf, Retrieved April 15, 2013.

[24] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," Journal of Electronic Imaging, vol. 13, no. 1, pp. 146–165, 2004.

[25] N. Otsu, "A threshold selection method from gray level histogram," IEEE Transactions on System, Man, Cybernetics, vol. 19, no. 1, pp. 62–66, January 1978.

[26] Y. Solihin and C. Leedham, "Integral ratio: A new class of global thresholding techniques for handwriting images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 8, pp. 761–768, August 1999.

[27] B. Gatos, I. Pratikakis, and S. J. Perantonis, "Adaptive degraded document image binarization," Pattern Recognition, vol. 39, no. 3, pp. 317–327, March 2006.

[28] I. K. Kim, D. W. Jung, and R. H. Park, "Document image binarization based on topographic analysis using a waterfall model," Pattern Recognition, vol. 35, pp. 141–150, 2002.

[29] J. Sauvola and M. Pietikainen, "Adaptive document image binarization," Pattern Recognition, vol. 33, no. 2, pp. 225–236, January 2000.

[30] W.Niblack, An Introduction to Digital Image Processing.Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

[31] Kulyukin, V., Kutiyanawala, A., and Zaman, T. (2012). Eyes-Free Barcode Detection on Smartphones with Niblack's Binarization and Support Vector Machines . In Proceedings of the 16-th International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV 2012), Vol. I, pp. 284-290, CSREA Press, July 16-19, 2012, Las Vegas, Nevada, USA, (pdf); ISBN: 1-60132-223-2, 1-60132-224-0.

[32] Leptonica Image Processing Library, http://www.leptonica.com/binarization.html#GLOBAL, Retrieved April 15, 2013.

[33] Apache Lucene, http://lucene.apache.org/core/, Retrieved April 15, 2013.

[34] Apache Lucene Spell Checker, http://lucene.apache.org/core/3_6_2/api/all/index.html, Retrieved April 15, 2013.

[35] Damerau, F.J. 1964. A technique for computer detection and correction of spelling errors. In Communications of ACM, 7(3):171-176.

[36] Vladimir I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Doklady Akademii Nauk SSSR, 163(4):845-848, 1965 (Russian). English translation in Soviet Physics Doklady, 10(8):707-710, 1966.

[37] Karen Kukich, Techniques for automatically correcting words in text, ACM Computing Surveys (CSUR), v.24 n.4, p.377-439, Dec. 1992

[38] Niklas, Kai, et al. Unsupervised post-correction of ocr errors. Diss. Master's thesis, Leibniz Universität Hannover, 2010.

[39] Farag Ahmed, Ernesto William De Luca, and Andreas N¨urnberger.MultiSpell: an N-Gram Based Language-Independent Spell Checker.In Proceedings of Eighth International Conference on Intelligent TextProcessing and Computational Linguistics (CICLing-2007), MexicoCity, Mexico, 2007.

[40] Grzegorz Kondrak, "N-gram similarity and distance". Proceedings of the Twelfth International Conference on String Processing and Information Retrieval (SPIRE 2005), pp. 115-126, Buenos Aires, Argentina, November 2005.

[41] Edward Fredkin, Trie memory, Communications of the ACM, v.3 n.9, p.490-499, September 1960

[42] Android, http://www.android.com/, Retrieved April 15, 2013.

[43] Apache Web Server Project, http://httpd.apache.org/, Retrieved April 15, 2013.

[44] Imagemagick, http://www.imagemagick.org/script/index.php, Retrieved April 15, 2013.