OFFLINE HANDWRITTEN CHARACTER RECOGNITION USING INTELLIGENT CHARACTER RECOGNITION IN ANDROID

By

Lim, Kimberly D.

Serrano, Geno Alfred A.

SCHOOL OF ARTS AND SCIENCES

ATENEO DE DAVAOUNIVERSITY

OCTOBER 2015

OFFLINE HANDWRITTEN CHARACTER RECOGNITION USING INTELLIGENT CHARACTER RECOGNITION IN ANDROID

An Independent Study

Presented to

The Faculty of the Computer Studies Cluster

Ateneo de Davao University

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Information Technology

By

Lim, Kimberly D.

Serrano, Geno Alfred A.

SCHOOL OF ARTS AND SCIENCES

ATENEO DE DAVAOUNIVERSITY

OCTOBER 2015

ACKNOWLEDGMENTS

First of all, the researchers are grateful to the Almighty God for guiding us through the years and giving us the strength to continue moving forward.

The researchers wish to express our sincere thanks to Mr. Bernie Jereza, the adviser, for his guidance and support as throughout the whole journey of this research. For without him, this research may have been given up halfway.

The researchers would also like to express their gratitude to Ms. Ma. Teresa Quindoy, the IT320 class adviser and panel chair, for her constant reminders and supervision that pushes the research to do more. If it weren't for her, a lot of valuable time could have been lost to procrastinating. To Mr. Patrick Paasa, panel member, the researchers are truly thankful for your support and suggestions throughout the project.

The researchers are thankful for the help and support of a fellow classmate, Mr. Peter Andrei Abellana. His kindness and knowledge has led this research to finish the technical output on time which the researchers are truly grateful for.

The research would like to take this opportunity to record its sincere thanks to all the faculty members of the Computer Studies Cluster for helping prepare the proponents for this research. The researchers also thank their parents for their unending encouragement and support.

This research would also like to place on record, the sense of gratitude to one and all who, directly or indirectly, have lent their helping hand in this research.

TABLE OF CONTENTS

LIST OF FIGURES

6.2.7.1. Snippet of code for using the Tess-two (a fork of tesseract tools for android).

6.2.7.2. Output File(text file)

LIST OF TABLES

Offline Handwritten Character Recognition using Intelligent Character Recognition in Android

LIM, KIMBERLY D.,Ateneo de Davao University
SERRANO, GENO ALFRED A.,Ateneo de Davao University

Intelligent character recognition is an advance optical character recognition or better known as handwritten character recognition. This research aims to recognize handwritten text extracted from a digital image to an editable text file in an Android device. The application would recognize a text image through Tesseract, an open source OCR engine, trained only to recognize handwritten characters.

General Terms: Intelligent Character Recognition (ICR), Optical Character Recognition, Tesseract, Adaptive thresholding, connected component labeling

Additional Key Words and Phrases: Adaptive Thresholding, noise reduction, image preprocessing

1. INTRODUCTION

1.1 Background of the Study

Smartphones are becoming more and more prominent in today's society. With its portability and convenience, applications, filling users wants and needs, starts to pop up in large masses. We are now in an era where almost everything is computerized. Smartphones or android phones were made to make things easier for us and eventually have become an essential part of the lives of people, specifically students.It may be due to this that people tend to be lazier than usual. An example would be assignments, reports, drafts of all sorts, and other documents of the likes written on paper would eventually be found tiring to type on computers especially when the written document is relatively long.

As a student, the idea of not all hardcopy has a softcopy is much known. Furthermore, writing drafts on paper before typing it in the computer is quite usual especially those who would be working the final draft at home or in an internet cafe.

Offline handwritten text recognition application involves the reception and interpretation of handwritten text/characters through a machine, specifically an android device. This is possible by using optical character recognition (OCR), specifically Intelligent Character Recognition (ICR), to recognize handwritten texts in images. It had been said that offline handwriting recognition is generally observed to be more difficult than the online handwriting recognition, given that features can be extracted from the pen trajectory and the image itself.

According to Jesse Hansen, "the goal of Optical Character Recognition is to classify optical patterns (often contained in digital images) corresponding to the alphanumeric or other characters". Its process consist of several steps which includes segmentation, feature extraction, and classification.

To put bluntly, android phones are handheld computers with phone services. Given that most students have android phones, implementing the application in an android device would make it easily accessible for all to use anytime and anywhere.

1.2 Problem Statement

Handwritten recognition systems involves the reception and interpretation of handwritten characters by a machine, but due to the variation of shape of handwritings, it becomes difficult for the machine to acquire its features for comparison. Despite that, recent research of ICR have increased handwritten recognition's accuracy.

The main problem of the study is how to translate image of handwritten text to a typewritten text.

The specific problems of the study are as follows:

(1) How to implement Intelligent Character Recognition (ICR)?
(2) How to implement adaptive thresholding?
(3) What are the appropriate algorithms to properly support ICR?
(4) What types of features are to be extracted from the image?
(5) How to implement the application to an android device?
(6) How to achieve accurate and fast recognition rate?

1.3 Objectives

The main objective of the study is to develop an android application that would translate image of handwritten text to typewritten text.

The specific objectives of the study are as follows:

1 To implement Intelligent Character Recognition
2 To implement adaptive thresholding
3 To determine the appropriate algorithms to properly support ICR
4 To determine the types of features to be extracted from the image
5 To implement the application to an android device.
6 To achieve accurate and fast recognition rate.

1.4 Significance of the Study

It had been said that recognizing handwritten text is more challenging than recognizing machine printed text given that every person has their own handwriting style. However, pursuing the topic would then lead to the continuation of the study of recognizing handwritten text which would eventually become useful in future projects. Furthermore, implementing the study in an android phone would increase its accessibility, for anyone would rather carry an android phone than having to bring laptops to experience the application.

1.5 Scope and Limitations

Although our main objective is to recognize handwritten text and turning it into machine printed text, limitations do exist such as text written in cursive form, slanted, or any type of illegible hand

writing would not be properly recognized by the application. Our project only exist to recognize characters within the English alphabet along with digits; special characters/symbols are excluded.

To avoid misrecognition of letters and digits due to looking alike when written by hand (e.g. the letter “O” and the digit “0”), the research decided to create some rules which includes:

(a) when writing the number ‘0’, it should be written with a slash or ‘/’ in the middle or ‘0’,
(b) the number ‘5’ should be written properly to avoid being classified as the upper case ‘S’,
(c) the number ‘1’ and a upper case ‘I’ will not be recognized if written as a straight line or ‘|’, this is to avoid being confused with the lower case ‘l’,
(d) the number ‘9’ should not be written with a tail to avoid confusion with the lower case ‘g’,
(e) the lower case ‘q’ should not be written as ‘q’ as to avoid confusion with the non-tailed ‘9’,
(f) the lowercase ‘a’ should be written as ‘a’,
(g) the upper case ‘J’ should be written as ‘J’, and
(h) the lower case ‘z’ should be written with a ‘-‘ in the middle as to not be recognized as its upper case counterpart or ‘Z’.

2. REVIEW OF RELATED LITERATURE

2.1 Adaptive Thresholding for DigitalDesk [3]

This report describes the thresholding problem which must be overcome by DigitalDesk applications followed by the adaptive thresholding algorithm that they decided to use (i.e. quick adaptive thresholding).

2.1.1 DigitalDesk

DigitalDesk is a suite of tools designed to enhance the learning and assessment processes of education institutions. Their modularized components allow users to build a suite of tools from test center management, content authoring to diverse methods of content and test delivery.

2.1.2 Adaptive Thresholding based on Wall’s algorithm

One technique for calculating a threshold that varies across the image according to background illumination was developed by R.J. Wall. It is described as breaking up the image into smaller tiles and calculates a histogram for each tile. Based on the peaks of these histograms, a threshold is chosen for each tile. Then every point in the image is assigned a threshold by interpolating between the values chosen for each tile.

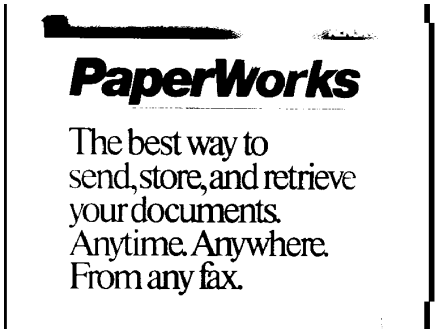


Fig. 2.2.1. Result of adaptive thresholding based on wall's algorithm

The image was divided into 9 (3x3) tiles and for each threshold was selected 20% below the peak. The result is much better than global thresholding, but because it requires more than one pass through the image, it is quite slow.

2.1.3 Quick Adaptive Thresholding

The basic idea for this is to run through the image while calculating a moving average of the last *s* pixel seen. When the value of a pixel is significantly lower than this average it is set to black, otherwise it is left white. Only one pass through the image is necessary, unlike most complex algorithms that require more passes through the image and take even longer to run, and the algorithm is simple enough to be implemented in hardware.

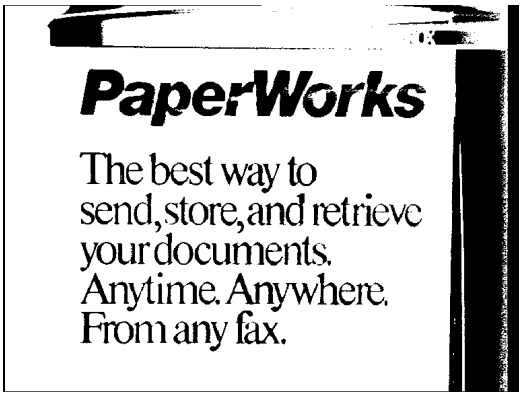


Fig. 2.2.2. Moving average scanning from left to right

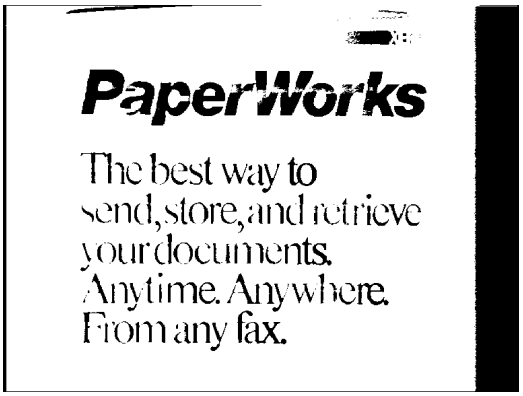


Fig. 2.2.3. Moving average scanning from right to left

2.2 ABBYY Mobile OCR Engine [6]

ABBYY mobile OCR engine enables developers to integrate OCR into mobile and small-footprint applications. This software development kit enables images and photographs to be transformed into searchable and editable document formats and supports all of the most popular mobile patforms and devices.

ABBYY has four (4) steps: image import and processing, document analysis, optical character recognition (OCR), and result processing. During step one, there is the automatic image skew correction function along with orientation detection and binarization. Step two is the removal of noise that is joined with the detection of characters and words. Once the two preparatory steps are done, the actual recognition commence followed by the last step wherein the recognition results can be processed and exported.

2.3 Text Fairy [7]

Text Fairy is an android OCR application available in play store. It is a simple kind of OCR that relies on Tesseract for its OCR processing and added additional feature like turning the output of Tesseract into a PDF file.

3. METHODOLOGY

An overall look of the system is the user taking an image of handwritten text with his android phone wherein it would be processed. Process includes ICR being performed which going through image preprocessing methods such as image denoising and deskewing, and undergo adaptive thresholding before being fed to Tesseract for recognition.

The input image could be any hand written text on a sheet of paper. This image would then be scanned by the android device's camera, digitizing the analog document. Texts regions are then recognize to be preprocessed, and then recognized.

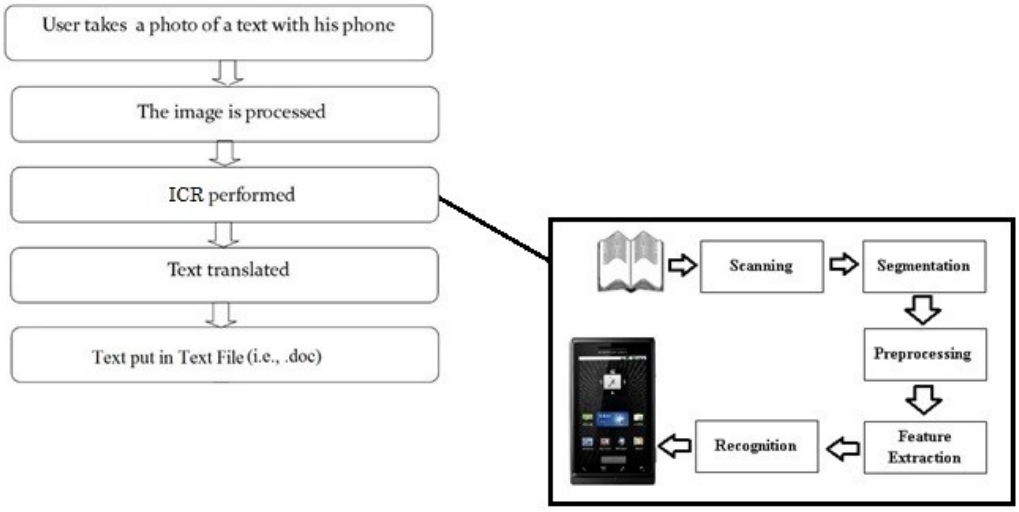


Fig. 3.1. Overview

Since taking there are many image impurities (e.g. noises) upon taking a picture, the research used three types of image preprocessing which includes noise reduction, skew normalization, and thresholding.

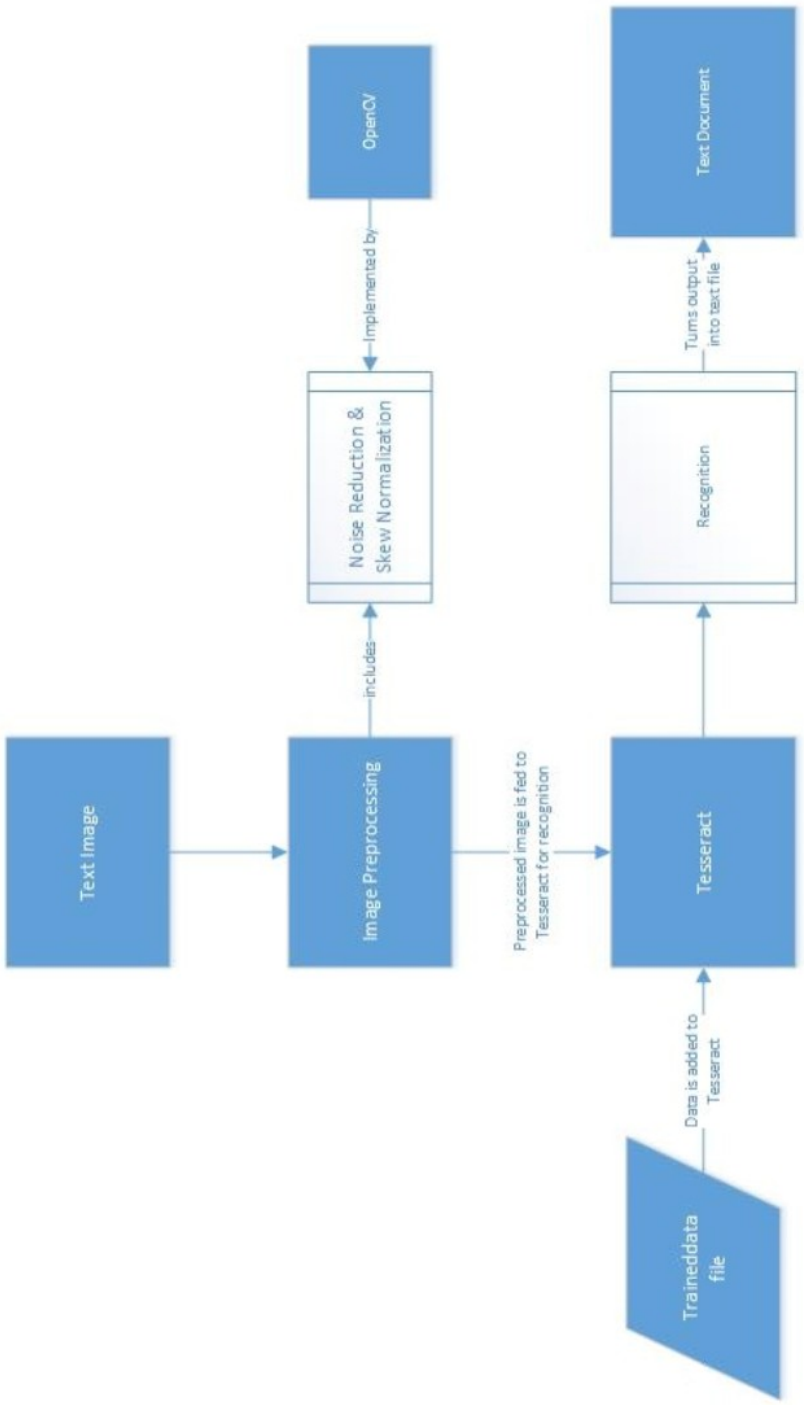
Using Gaussian blur, a type of noise reduction, it will remove unnecessary pixels in the picture that could cause misrecognition by bluring. The research implemented this using the open source OpenCV.

Thresholding is important for the segmentation process and for this research, adaptive thresholding was used. Normal/Global thresholding is also good, however, it creates new noises when the lighting during image capturing is imbalanced. Adaptive thresholding solves that problem when using the global thresholding. Like Gaussian blur, adaptive thresholding was also implemented through OpenCV.

Since the research uses a photo taken from a mobile device, there are times that the picture comes out a slanted. Skew normalization should fix this problem. Skew normalization calculates and fix the skewness of the picture making it straight. There are many ways to do skew normalization, but the research decides to use the projection profile analysis which is a simple and straight forward type of skewing algorithm.

Once the image undergoes these methods, Tesseract, implemented in android, would now recognize the characters one by one using a traineddata file that was created manually through Tesseract, using the desktop (See in discussions—training process).

4. CONCEPTUAL FRAMEWORK


Text Image
Image Preprocessing
includes
Noise Reduction & Skew Normalization
Implemented by
OpenCV
Preprocessed image is fed to Tesseract for recognition
Tesseract
Recognition
Turns output into text file
Text Document
Data is added to Tesseract
Traineddata file

The conceptual framework shows two main process in the research: creation of the traineddata (done in a desktop) and the application's framework (done in Android). The desktop side should be processed first for the output of that side is the traineddata file which would then be imported to the Android side as the data for recognition.

The training image of tif/tiff type would be fed in Tesseract to create a box file for the image. The box file is much like the entire training process wherein the research would properly label each character here. Tesseract is only a command line so to edit a box file, a 3rd party program is used, which in our case, the jTessBoxEditor. Once the box file is properly edited, the training process begins followed by the generation of the unicharset, and the clustering and creating the normproto file. Once all the files are created, Tesseract would then put all of them together to create the traineddata file.

In the application side, the text image (i.e. the image to be recognized) would undergo image preprocessing to remove imperfection. The preprocessing includes the noise reduction and skew normalization, done through OpenCV, which is important for achieving high accuracy. Once the image is clear of imperfection, it would then be fed to Tesseract for it to recognize the characters using its own feature extraction and classifier based on the traineddata that the researchers have made beforehand. Once Tesseract recognized the characters, its output would then be added in a text file.

5. THEORETICAL BACKGROUND

5.1 Intelligent Character Recognition

The goal of Optical Character Recognition (OCR) is to classify optical patters, often contained in digital image, corresponding to alphanumeric or other characters. Optical character recognition can be classified to four (4) types: optical character recognition, optical word recognition, intelligente character recognition, and intelligent word recognition. Intelligent character recognition aims to recognize handwritten characters which is outside the scope of optical character recognition.

5.2 Tesseract [1] [2]

Tesseract is an OCR engine and is not a fully featured program. 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 JTessboxEditor. Without integration into programs such as these, Tesseract has no page layout analysis, no output formatting and no graphical user interface (GUI).

Tesseract uses different algorithms most, however, are algorithms when recognizing characters like algorithms for detecting proportional and proportional words (proportional word is a word where all letters are the same width), and algorithms for chopping joined characters and for associating broken characters. It also has linguistic analysis to identify the most likely word formed by a cluster of characters. Tesseract also uses two character classifiers: a static classifier, and an adaptive classifier which employs training data, and which is better at distinguishing between upper and lower cases.

5.3 OpenCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision. The library is cross-platform and free for use under the open-source BSD license. OpenCV was designed for computational efficiency. Written in optimized C/C++, the library can take advantage of multi-core processing. The library has more than 2500 optimized algorithm, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

Algorithms/methods used in the research includes grayscaling the image, the use of Gaussian blur to remove noise, and the adaptive thresholding.

5.4 Image Preprocessing

5.4.1 Noise Reduction

5.4.1.1 Gaussian Blur

Noises, like speckle, skew, and blur, could be remove by some calibration techniques if the model for it is available. However, modeling the noise is not possible in most of the application. The visual effect of this blurring technique is a smooth blur resembling that of a viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination.

5.4.2 Skew Normalization

5.4.2.1 Projection Profile Analysis [5]

Skew normalization based on projection profile is a simple and straight forward approach. Here, pixel values in each row of the image are extracted and finally by summing these values will give a column vector.

5.4.2.2 Hough Transform [8]

The Hough Transform is a powerful global method for detecting edges. It transforms between the Cartesian space and a parameter space in which a straight line (or other boundary formulation) can be defined. The advantages of the Hough transform is that the pixels lying on one line need not all be contiguous. This can be very useful when trying to detect lines with short breaks in them due to noise, or when the objects are partially occluded. The disadvantage is that it can give misleading results when objects happen to be aligned by chance. This shows another disadvantage which is that the detected lines are infinite lines rather than finite lines with defined end points.

5.4.2.3 Nearest Neighbor Clustering [9]

Similar to the k-nearest neighbor classifier in supervised learning, this algorithm can be seen as a general baseline algorithm to minimize arbitrary clustering objective functions.

5.5 Adaptive Thresholding [3]

According to Wellner, 'an ideal adaptive thresholding algorithm would produce the same result when applied to an unevenly lit page as a global thresholding algorithm would produce when applied to a perfectly evenly lit page.'

5.6 Android Device [4]

The target for this research is the Android OS platform as it is open source and is most commonly used in most of the devices in the market based on statistical sales reports of mobile operating systems (Statista, 2013). As of the data collected during a 7-day period ending on March 2, 2015, it is shown that android version codename: Jelly Bean (ver. 4.1.x, 4.2.x, and 4.3), and Kitkat (ver. 4.4) has the most number of distribution and users (Android Developers, 2015).

Specifications of the smartphone would vary as the processing power is important for the proposed application to work. At most as of the current, smartphones have at least 1.2GHz processing speed on average and at least 8 mega pixels (MP) on average for digital camera quality based on reports on smartphones (FindTheBest, 2015)

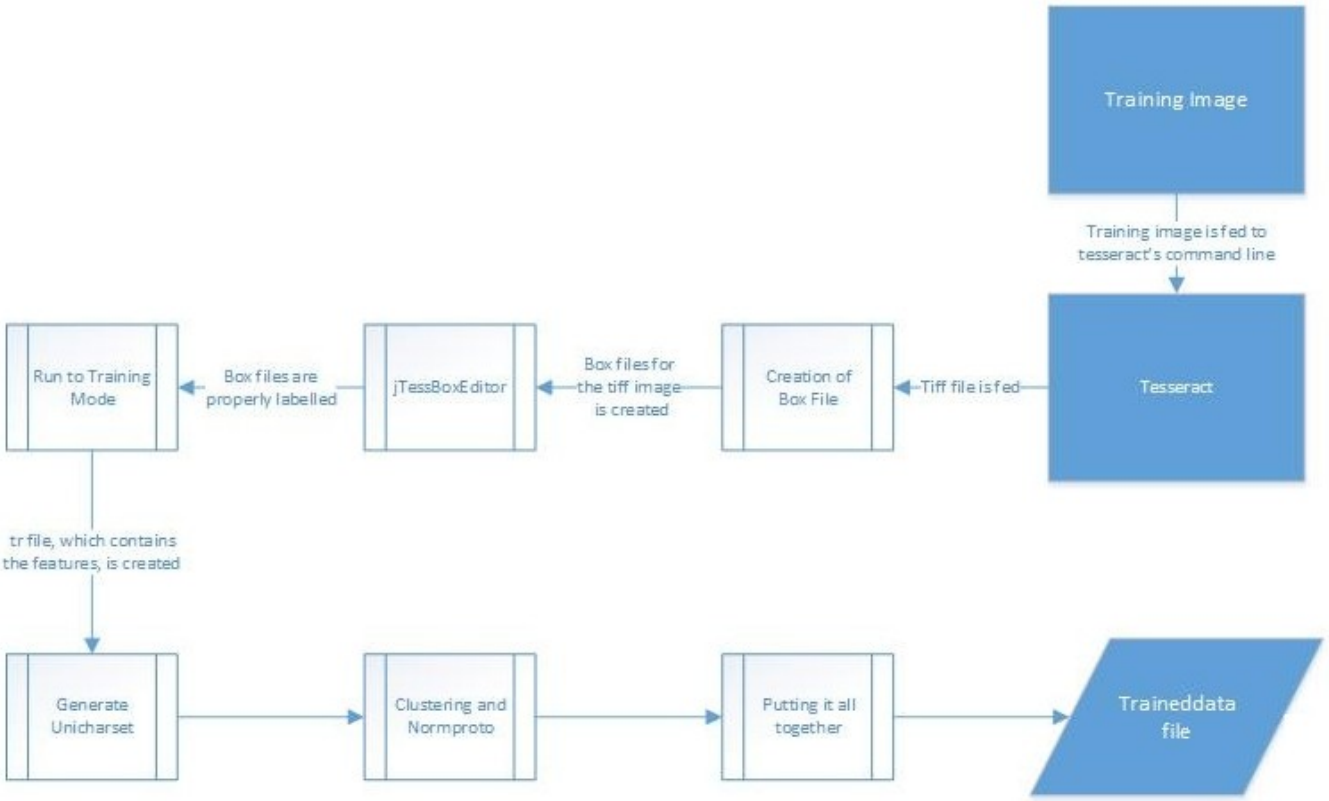
6. DISCUSSIONS

6.1 Desktop Implementation

The training process for Tesseract is done in the desktop command line or through a 3rd party program.

6.1.1 Creating Tesseract's Trained data

There are two ways to create Tesseract's traineddata, using a program for training data (e.g. SunnyPage) or the manual way. Using a program made for training data is a lot less complicated than creating the data manually through Tesseract since it usually just involves inputting an image, preferably a tif/tiff image, clicking through the boxes, and classifying them to their correct label. The manual way isn't really all that difficult either, but creating the traineddata file using the manual way means to use Tesseract's command line and that one has to fully understand the commands in Tesseract along with looking for programs that would fill in the gap in Tesseract (i.e. editing the box files).



6.1.2 **Training Process**

Fig. 6.1.2.1. Tesseract Training Process

Tesseract uses a traineddata file that acts as a database for its classification. There are two ways to acquire this file: through another program or creating it manually through Tesseract.

The research originally used the program SunnyPage in training our training image for about two months only to realize that it wasn't as efficient as the initial assumption. The research eventually turned to the manual way.

Tesseract already comes with traineddata file for many languages, however, all of those are rendered useless for those were data files to recognize machine printed text which doesn't match well with handwritten characters. In creating the traineddata file manually, the research would use Tesseract on the command line and since Tesseract is only an engine, the research used a program called jTessBoxEditor for the classification of the box files.

```
tesseract eng.timesitalic.exp0.tif eng.timesitalic.exp0 batch.nochop makebox
```

Fig 6.1.2.2 command for making box files from your training images

To generate the training files for a specific user, there's a need to prepare the box files for each training images using the following command above Fig 6.1.2.2.

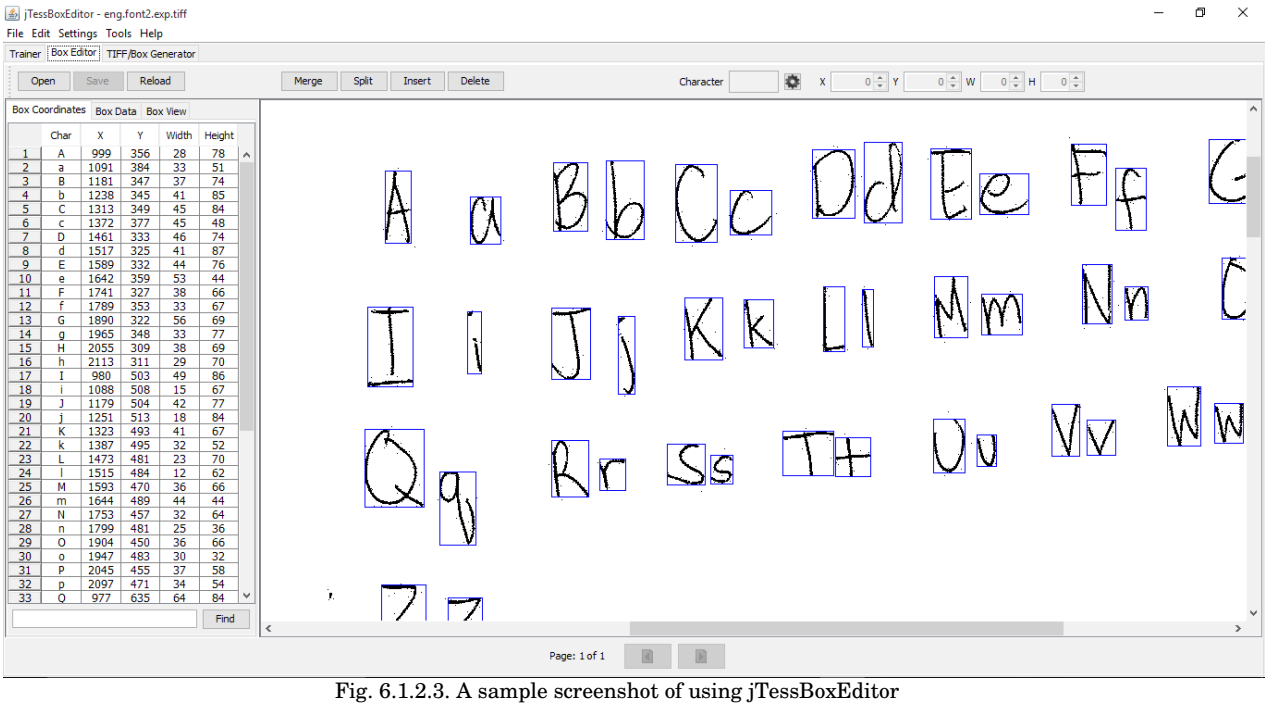


Fig. 6.1.2.3. A sample screenshot of using jTessBoxEditor

The research used jTessBoxEditor to edit the box data as shown in Fig. 6.1.2.3. For training a new language set for any user, the researchers have to put in the effort to get one good box file for a handwritten document page, run the rest of the training process, discussed below, to create a new language set.

```
tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num] box.train
```

Fig. 6.1.2.4 command to run tesseract in training mode

After the box files are edited, it is fed the box files to tesseract and the output of this step is fontfile.tr which contains the features of each character of the training page. The character shape features can be clustered using the mftraining and cntraining programs

```
unicharset_extractor lang.fontname.exp0.box lang.fontname.exp1.box ...
```

Fig. 6.1.2.5. command to generate unicharset data file

Tesseract needs to know the set of possible characters the command above is to generate unicharset data file from the box files generated before.

The output fontfile.tr which contains the features of each character of the training page. The character shape features can be clustered using the mftraining and cntraining programs:

```
shapeclustering -F font_properties -U unicharset eng.timesitalic.exp0.tr
mftraining -F font_properties -U unicharset -O eng.unicharset eng.timesitalic.exp0.tr
```

Fig. 6.1.2.6. command for clustering

```
cntraining lang.fontname.exp0.tr lang.fontname.exp1.tr ...
```

Fig. 6.1.2.7 command for generating the normproto data file

This will output three data files: inttemp, pffmtable and shapetable. Lastly the last file you need is normproto data file as shown in Fig. 6.1.2.7.

```
combine_tessdata lang.
```

Fig. 6.1.2.8. putting it all together

In Fig 6.1.2.8 the command will generate the trained data which will then be used as database by Tesseract Engine.

Once the traineddata is created, it would then be imported to the application for Tesseract to be used during recognition.

6.2 Android Implementation

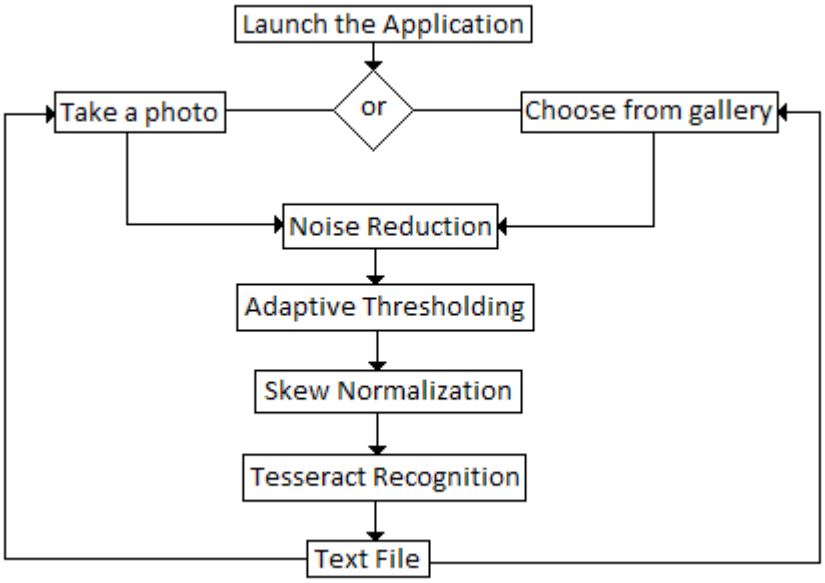


Fig. 6.2 Program flow for the Android Application

For image preprocessing, the researcher's main focus would be eliminating or lessen the imperfections of the image text that could cause affect the image recognition negatively. This would include the image noises and the image skewness. In tackling these imperfections, the researchers decided to use noise reduction and normalization for better character recognition.

6.2.1 Tesseract in Android

A tesseract library has a previously built android application project called *Tess-two* project but this project doesn't support build for gradle android projects which Android Studio requires. So to integrate *Tess-two* with gradle.

- create a libraries folder under project's main directory.
- Then move the entire *Tess-two* directory into the libraries folder.
- Delete libs folder inside the *Tess-two* directory then create **build.gradle** file in the *Tess-two* directory, edit **settings.gradle** file according to your application needs.
- Lastly, is to sync the project in Android Studio and add *Tess-two* library as **Module Dependency** to the main project.

6.2.2 OpenCV in Android

There is already a downloadable OpenCV library for Android. Below are the steps the research did to use OpenCV library in Android Studio.

- Import OpenCV to Android Studio: From *File -> New -> Import Module*, choose *sdk/java* folder in the extracted zipped file of OpenCV.

- Update **build.gradle:** just like in Tesseract, there's a need to update this according to the application needs specifics will be (**compileSdkVersion**, **buildToolsVersion**, **minSdkVersion** and **targetSdkVersion**).
- Then add the library as **Module Dependency,** and copy **libs** folder inside the OpenCV extracted zipped file to Android Studio *app/src/main* and rename it to **jniLibs.**
- To run OpenCV in your application the android phone needed the OpenCV Manager installed in order to run OpenCV functions.

6.2.3 Methods of Choosing a Text Image

```
final Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);

startActivityForResult(intent, 0);
```

Fig. 6.2.3.1. Snippet for image capturing

In the figure above, the research used and Intent *MediaStore.ACTION_IMAGE_CAPTURE* to open the camera for image capturing. Then *MediaStore.EXTRA_OUTPUT* would save the captured image to a directory that the researchers want it to be saved.

```
File pictureDirectory = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);

String pictureDirectoryPath = pictureDirectory.getPath();
Uri data = Uri.parse(pictureDirectoryPath);
photoPickerIntent.setDataAndType(data, "image/*");
startActivityForResult(photoPickerIntent, IMAGE_GALLERY_REQUEST);
```

Fig. 6.2.3.2. Snippet for taking a photo from the phone's gallery

The *pictureDirectory* variable receives the directory where the pictures are currently located. Then *data* variable will access by reference to the pictures located in the *pictureDirectory*. Invoking the Image Gallery through *photoPickerIntent* and *startActivityForResult*.

```
InputStream inputStream = getContentResolver().openInputStream(data.getData());
FileOutputStream fileOutputStream = new FileOutputStream(mFileTemp);
```

Fig. 6.2.3.3. Snippet for getting the image

After getting the pictures the selected image will be then in the variable *mFileTemp* a string which contains the directory or the full path of the image.

6.2.4 Image Preprocessing

For image preprocessing, the researcher's main focus would be eliminating or lessen the imperfections of the image text that could cause affect the image recognition negatively. This would include the image noises and the image skewness. In tackling these imperfections, the researchers decided to use noise reduction and normalization for better character recognition.

One the said imperfection had been fixed, the application of thresholding would extract the foreground, which are characters, from the rather monotonic background or gray scale wherein it is equal to the binarization of its real-valued data.

The research decided to use an open source library known as OpenCV to apply the image preprocessing methods that the research intend to use.

OpenCV is released under a BSD license and hence it's free for both academic and commercial use and is also cross-platfrom. In the early days of OpenCV, the goals of the project were described as (1) advance vision research by providing not only open but also optimized code for basic vision infrastructure, (2) disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable, and (3) advance vision-based commercial applications by making portable, performance-optimized code available for free.

6.2.4.1 **Noise Reduction**

Noises in pictures causes disconnected line segments, bumps and gaps in lines, filled loops etc. The distortion which includes local variations, rounding of corners, dilations and erosion, is also a problem. It is necessary to eliminate these imperfections prior to the character recognition.

6.2.4.2 **Normalization**

Normalization aims to remove all types of variation during the writing and obtain standardized data. The following are some basic methods for normalization.

6.2.4.2.1 Skew Normalization

Due to the inaccuracies in the scanning process and writing style, the writing may be slightly tilted within the image. This could affect the effectiveness of the later algorithms and therefore, should be detected and corrected. Methods in detecting the amount skew includes using the projection profile of the image, using the Hough transform, and a form of nearest-neighbor clustering. After the detection, the character or word is translated to the origin, rotated until the baseline is horizontal and re-translated back into the display screen space.

The research noticed that upon using the research's chosen skewing algorithm, it generates new 'noises' which is most of the time seen before the first line and after then last line.

6.2.5 **Thresholding**

Thresholding is the extracting of the foreground, which are characters in our case, from the rather monotonic background. In a gray scale image, this is equal to the binarization of its real-valued data. In order to reduce storage requirements and to increase processing speed, it is often desirable to represent grey scale or color images as binary images by picking some threshold value for everything

above that value is set to 1 and everything below is to 0. There are 2 categories of thresholding: Global and Adaptive. Global thresholding picks one threshold value for the entire document image.

6.2.5.1 Adaptive Thresholding

Fixed thresholding often fails if the illumination varies spatially in the image. The common solution to this is adaptive thresholding. Adaptive thresholding is a form of thresholding that takes into account spatial variations in illumination. The difference of adaptive thresholding and fixed thresholding is that a different threshold value is computed for each pixel in the image. This technique provides more robustness to change in illumination.

6.2.6 **Implementation of Image Preprocessing and Adaptive Thresholding**

Tesseract has its own preprocessing methods using Leptonica library though the specifics weren't stated, however, in Tesseract's page, it states numerous preprocessing methods that could increase accuracy such as noise removal, binarization, oriented/skew normalization, and borders.

Through an open source library OpenCV, the research were able to implement three out of four preprocessing recommendations of Tesseract (i.e. noise removal, binarization/thresholding, and skew normalization).

```
try {
    Mat tmp = new Mat();
    d_image = BitmapFactory.decodeFile(mFileTemp.getPath());
    Utils.bitmapToMat(d_image, tmp);
    Imgproc.cvtColor(tmp, tmp, Imgproc.COLOR_RGB2GRAY);
    Imgproc.GaussianBlur(tmp, tmp, new Size(3, 3), 0, 0);
    Imgproc.adaptiveThreshold(tmp, tmp, 255, Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C, Imgproc.THRESH_BINARY, 5, 4);
    Imgproc.threshold(tmp, tmp, 100, 255, Imgproc.THRESH_OTSU);
    Imgproc.GaussianBlur(tmp, tmp, new Size(3, 3), 0, 0);

    double skew = compute_skew(tmp);
    if(skew != 0) {
        tmp = deskew_image(tmp, skew);
    }
    Utils.matToBitmap(tmp, d_image);
    image = d_image.copy(Bitmap.Config.ARGB_8888, true);
    imgPicture.setImageBitmap(image);
```

Fig. 6.2.6.1 Image Preprocessing using open-cv

There's a need to convert the image into matrix by using the *Utils.bitmapToMat()*. Then process the image using opencv tools the Imgproc library. First is to convert the picture to grayscale by using *Imgproc.COLOR_BGR2GRAY* function. Then, denoising the image using Gaussian Blur algorithm also in the opencv tools library Imgproc. Using the adaptive thresholding in opencv with the adaptive method ADAPTIVE_THRESH_GAUSSIAN_C in which the threshold value is the weighted sum of neighborhood values and the applying Otsu thresholding to the image. Lastly would be to deskew the image.

```
Mat lines = new Mat();
double angle = 0;
Mat tmp = new Mat();
img.copyTo(tmp);
Bitmap test_image = null;
Core.bitwise_not(img, tmp);
try {
    Imgproc.HoughLinesP(tmp, lines, 1, Math.PI / 180, 65, 40, size.width/2.f);
    Mat disp_lines = new Mat(size, CvType.CV_8UC1, new Scalar(0, 0, 0));
    for (int x = 0; x < lines.cols(); x++)
    {
        double[] vec = lines.get(0, x);
        double x1 = vec[0],
                y1 = vec[1],
                x2 = vec[2],
                y2 = vec[3];
        Point start = new Point(x1, y1);
        Point end = new Point(x2, y2);

        angle += Math.atan2(y2 - y1, x2 - x1);

        Imgproc.line(disp_lines, start, end, new Scalar(255,0,0), 3);
    }
    angle /= lines.cols();
    Bitmap bmp = Bitmap.createBitmap(disp_lines.cols(), disp_lines.rows(), Bitmap.Config.ARGB_8888);
    Utils.matToBitmap(disp_lines, bmp);
    bmp.getHeight();
}
catch (Exception e) {
    Log.e(TAG, "DESKEWING COMPUTATION ERROR");
}
return angle;
```

Fig. 6.2.6.2. Algorithm for computing skew angle in an image.

In computing the skew angle the image must be inverted the background will be black and the text will be white through the method in opencv *Core.bitwise_not()* then find some straight lines in the text. OpenCV has a method in finding the lines in the image called Hough transform then the image is scanned by a voting system to know the best lines identified. After getting the best line in the image, it shall compute now for the angle between each line and the horizontal line using Math.atan2 function and compute the mean angle of all the lines then return the value computed.

```
private static Mat deskew_image(Mat img, double skew) {
    //TODO DESKEW IMAGE
    Point center = new Point(img.width() / 2, img.height() / 2);
    Mat rotImage = Imgproc.getRotationMatrix2D(center, skew * 180 / Math.PI, 1.0);
    //1.0 means 100 % scale
    Imgproc.warpAffine(img, img, rotImage, img.size(), Imgproc.INTER_LINEAR + Imgproc.CV_WARP_FILL_OUTLIERS);
    return img;
}
```

Fig. 6.2.6.3. Image deskewing.