ANDROID APPLICATION FOR DEVANAGARI SCRIPT OCR

A Project Report

Submitted by

Shreyas Ambekar	**111603002**
Hrishikesh Budhwant	**111603008**
Pranit Gandhi	**111603017**

in partial fulfillment for the award of the degree
of

B.Tech Computer Engineering

Under the guidance of
Prof. Abhijit A.M.
College of Engineering, Pune

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY,
COLLEGE OF ENGINEERING, PUNE-5

June, 2020

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY, COLLEGE OF ENGINEERING, PUNE

CERTIFICATE

Certified that this project, titled "ANDROID APPLICATION FOR DEVANAGARI SCRIPT OCR" has been successfully completed by

Shreyas Ambekar	**111603002**
Hrishikesh Budhwant	**111603008**
Pranit Gandhi	**111603017**

and is approved for the partial fulfillment of the requirements for the degree of "B.Tech. Computer Engineering".

SIGNATURE
ABHIJIT A.M.
Project Guide
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

SIGNATURE
VAHIDA ATTAR
Head
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

Abstract

Main aim of the project is to have an Android application that can run OCR process on input images on the device itself(not on the server) for Devanagari script and use the latest OCR engine to do so.The application has considerable speed of execution and an acceptable accuracy of the output. A nice GUI with rich feature set has been offered with the app to reduce human effort while processing the images. In this project, Tesseract 4.0 OCR has been integrated with the Android application. There is always an option of creating an OCR engine from scratch, but an existing OCR engine has been used and the reasons for this choice are discussed in the Literature Survey part. A considerable number of digitized document images are poor in quality, varying in fonts, font sizes, scripts and styles. Due to this a considerable amount of effort has been put to improve the accuracy of the OCR process by carrying out the required image pre-processing tasks. Further the report contains explanation of the decision to develop the application in Java for Android while other popular options like React Native exist. The system requirements, System Design, Implementation details, results and future scope required for maintaining implementation have been discussed at length.

Contents

List of Figures iii

List of Tables iv

List of Abbreviations v

1 Introduction 1

2 Literature Survey 2

2.1 Use of existing OCR engine 2

2.2 Comparison of different OCR engines 2

2.3 Version 4.0 of Tesseract 4

2.4 Need of preprocessing the image 6

2.5 Need for setting the resolution based on text size 6

2.6 Preprocessing image on server side 7

2.7 Comparison of Different options for implementation – (Native apps and hybrid apps) 8

3 System Requirement Specification 10

3.1 Functional Requirements 10

3.2 System Requirements 10

3.3 Software Requirements 10

3.4 Non Functional Requirements 11

4 System Design & Architecture 12

4.1 Components in the application 12

4.2 Server side components 13

4.3 Image preprocessing functionalities 14

4.4 Use case diagram 14

5 Implementation 15

5.1 Client side 15

5.1.1 Client side implementation 15
5.1.2 User Interface of the application 16
5.2 Server side 21
5.2.1 Server side implementation 21
5.2.2 Snapshots of server side 21
5.2.3 Image preprocessing operations 24

6 Results **30**

7 Future Scope **37**

8 Conclusion **38**

9 References **39**

A Pseudocode **42**

List of Figures

1 Google trend analysis of OCR engines. 3
2 Accuracy drop for unprocessed images 6
3 Tesseract error rate vs font resolution 7
4 Component Diagram 12
5 Server-side components 13
6 Image Preprocessing process flow diagram 14
7 Use case diagram 14
8 Launcher Activity - Grid of documents 16
9 Selecting image from Gallery 17
10 Cropping the image to choose the text section 17
11 Dialog box asking user if they want to upload the image 18
12 'Loading Spinner' while uploading the image 18
13 Selecting the language/script of the text 19
14 Selecting the layout between Simple or Complex 19
15 Activity displayed while performing the OCR on the preprocessed image 20
16 Final result of OCR 20
17 Connecting to the AWS EC2 instance with the help of SSH 22
18 Status of Gunicorn WSGI 22
19 Status of Nginx Web Server 23

List of Tables

1 Comparison of OCR engines 4

2 Tesseract error rates for version 3 and 4 5

3 Error rates 30

List of Abbreviations

OCR	Optical Character Recognition
LSTM	Long Short Term Memory
MODI	Microsoft Office Document Imaging
WSGI	Web Server Gateway Interface
DPI	Dots Per Inch
OEM	OCR Engine Mode
PSM	Page Segmentation Mode
JNI	Java Native Interface
ndk	Native Development Kit
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
AMI	Amazon Machine Image

Chapter 1

Introduction

OCR(Optical Character Recognition) can be found in various activities of our daily life. It can improve any process that has to deal with text capture or image processing of printed text. OCR software has improved the ability to enter, store and use data by leaps and bounds. Many industries utilize OCR; data entry is a major component of banking, finance, education, healthcare, and government jobs. All of these fields contain forms and printed documents that are becoming increasingly digitized. To make the transfer from hard copy to digital format, papers need to be scanned and relevant information needs to be extracted. .

Most of the work carried out in document analysis and character recognition is on Latin based scripts. There is only a limited research made for the recognition of Indian languages based on Devanagari script. Compared to the European languages, OCR systems haven't yet been successful in recognizing Devanagari document images of varying fonts, scripts, quality, size and style. Devanagari script poses a lot of challenges like (1) Large number of vowels and consonants. (2) difficulty in developing language processing routines. (3) Complicated language constructs. .

In India, with many official languages using Devanagari Script, a software in which OCR technology will be used will greatly improve the data entry process and help to eliminate the burdens of using and managing paper and other traditional types of documents.

Chapter 2

Literature Survey

2.1 Use of existing OCR engine

To solve the problem of Devanagari script OCR, different steps involved in it were studied. There are different steps like scanning, preprocessing, segmentation, feature extraction, classification and post processing. There are many existing OCR engines(e.g. Tesseract, Google Cloud Vision) that provide good performance, so it is not necessary to build one from scratch. Building the OCR engine with acceptable accuracy and performance is a very complex and time consuming task. This motivated the use of existing OCR engines.

2.2 Comparison of different OCR engines

Various OCR engines and softwares were analyzed in order to choose one with acceptable accuracy as well as capable of efficient processing on a mobile device. Different applications with the OCR functionality were studied along with the OCR engines they use. The conclusion being, many applications run on PCs on Windows, Linux or both and very few applications developed for Android. With nearly everyone having an Android phone now-a-days and the number of Android smartphone users increasing day by day, the OCR application running on Android will be of great use.

Comparison was done based on various aspects discussed above. The OCR of the image should be done on the client side (on the mobile device itself)so that no internet is needed - the facility which most of the applications today avoid. Many existing applications send images to the server for processing. Many existing applications are proprietary and use paid APIs. Also a large number of applications exist for English OCR, but there are very few applications which solve the Devanagari Script OCR problem. Some appli-

cations provide support for Devanagari script, but they use old OCR engines which are less accurate and slow.

The following google trend analysis graph shows the recent trends of the use of different available OCR engines.

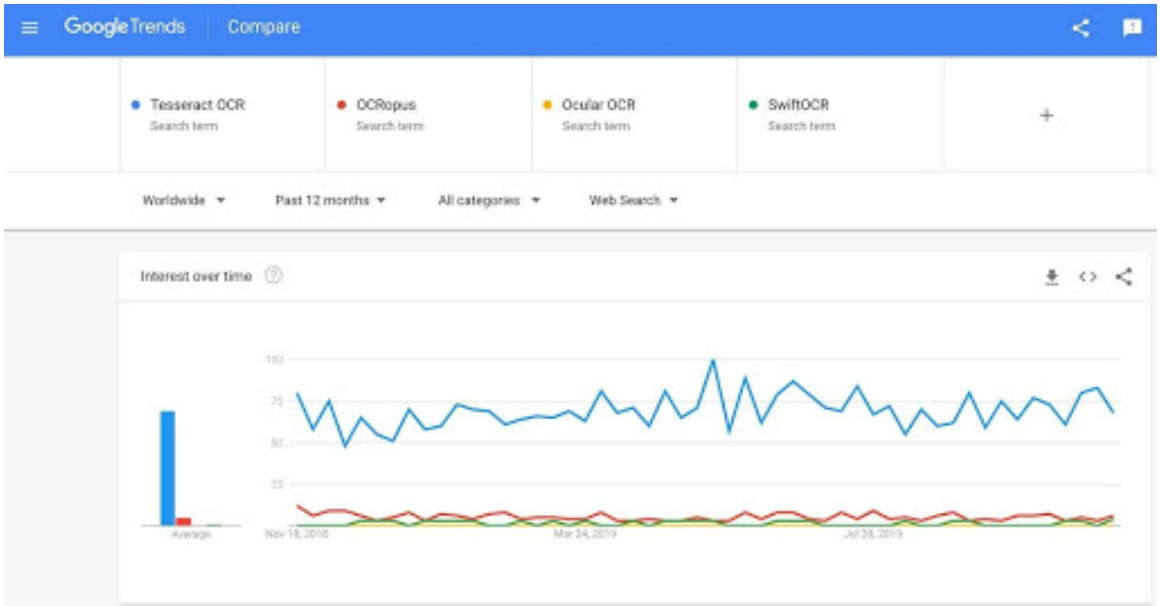


Figure 1: Google trend analysis of OCR engines.

Image downloaded from https://nanonets.com/blog/ocr-with-tesseract/

The table shows the comparison of all the OCR engines in short and all the softwares that have used these OCR engines. They are compared on the basis of various factors like the fonts supported, their license, the supported languages, client side/server side processing, operating systems supported, etc. The table shows the comparison of popular OCR engines and softwares.

Name	License	Online	Windows	Mac OS X	Linux	Languages	Fonts
Tesseract	Apache	No	Yes	Yes	Yes	100+	Any Printed Font
Google Cloud Vision	Free	Yes	Browser	Browser	Browser	200+	All Fonts
E-aksharayan	-	-	Yes	No	Yes	14	-
MS Office One Note 2007)	Proprietary	No	Yes	No	No	-	-
Smart Score	Proprietary	No	Yes	Yes	No	-	-
OCRopus	Apache	No	No	Yes	Yes	All languages using Latin Script	Normal Latin script and Fraktur (other scripts can be trained)
Puma.NET	BSD	No	Yes	No	No	28	Any printed font

Table 1: Comparison of OCR engines

Image downloaded from

https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software

2.3 Version 4.0 of Tesseract

Tesseract 4.0 uses a recurrent neural network(LSTM) based OCR engine focused on line recognition. It is faster than Tesseract 3.0 in both wall clock time(the time elapsed from the start of process as measured by a clock) and CPU time(the amount of time for which a CPU is busy processing instructions of a computer program). Tesseract 4.0 supports 3 sets of language models unlike Tesseract 3.0 which supports only one. These 3 sets are as explained below :

tessdata - It can be used with legacy as well as new LSTM based OCR engine. It is slightly less accurate than tessdata_best.

tessdata_best - It can only be used with Tesseract 4.0. It is the slowest but most accurate amongst the three.

tessdata_fast - It contains language models with a smaller neural network. Float to int conversion is done to reduce the size of the neural network. As a result, it is the fastest but least accurate amongst the three.

As Tesseract version 4.0 OCR engine had significant performance and accuracy gains with respect to Tesseract 3.0, it was selected for this application. Another notable point being Tesseract version 4.0 can be easily integrated by its wrapper library **Tesseract4Android.**

The following error rates(from the Tesseract official documentation) of OCR using base Tesseract(Tesseract version 3) and Tesseract(Tesseract version 4) LSTM were considered.

Accuracy Results (Indic Languages)

Lang	Truth Words	Char Error Rates			%Change	Word Error Rates			%Change
		Base	T-LSTM	LSTM+Dict		Base	T-LSTM	LSTM+Dict	
Bengali	16865	19.59	22.33	19.25	-1.74	42.79	42.23	39.13	-8.55
Gujarati	19874	45.7	22.38	18.35	-59.85	56.46	49.46	43.62	-22.74
Hindi	27539	14.05	13.56	11.68	-16.87	30.92	30.19	26.92	-12.94
Kannada	13673	31.55	13.57	12.12	-61.58	63.2	47.11	40.71	-35.59
Marathi	21486	29.04	12.07	9.18	-68.39	47.19	32.79	25.92	-45.07
Nepalese	20606	29.56	18.38	15.63	-47.12	50.19	42.39	36.59	-27.10
Panjabi	27651	46.23	19.89	16.1	-65.17	54.89	40.18	37.15	-32.32
Tamil	10033	26.1	10.09	9.3	-64.37	63.58	30.65	26.78	-57.88
Telugu	14133	31.47	31.51	27.8	-11.66	63.63	63.79	59.36	-6.71

Google

Table 2: Tesseract error rates for version 3 and 4

Image downloaded from https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/7Building%20a%20Multi-Lingual%20OCR%20Engine.pdf

2.4 Need of preprocessing the image

The accuracy of output of any Tesseract engine can be improved if the image is preprocessed before giving it as an input. Improved image quality makes it easy for the OCR engine to identify a character from its background. Image preprocessing leads to sharp character borders, higher contrast and lesser noise. Many applications lack this feature of preprocessing the image before it is given as an input to the OCR engine. So, the image preprocessing operations have been added to the application. The following graph shows the accuracy drops when images are given without processing as an input to the OCR engine.

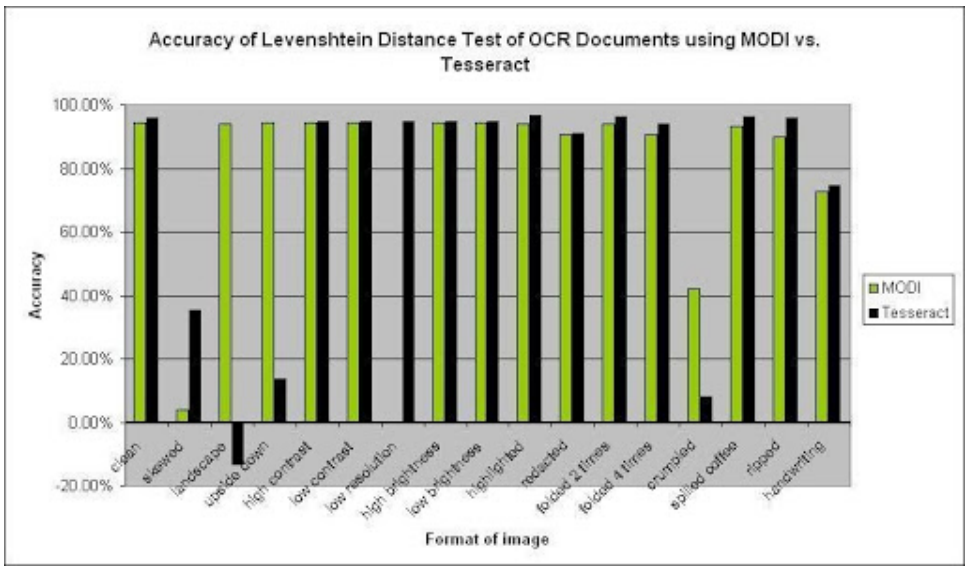


Figure 2: Accuracy drop for unprocessed images

Image downloaded from

http://discoverybrain.com/ediscovery/surprising-research-on-ocr-technologies/

2.5 Need for setting the resolution based on text size

The Tesseract OCR error rate is strongly correlated with the font resolution. The following graph confirms that Tesseract 4.0 is well suited for lower resolutions, but compared to Tesseract 3.0, it consistently performs better across a broader range of fonts.

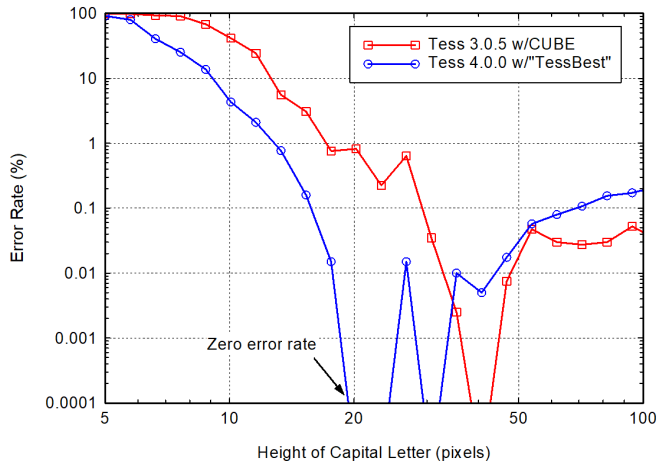


Figure 3: Tesseract error rate vs font resolution

Image downloaded from https://groups.google.com/forum/#!msg/tesseract-ocr/Wdh_JJwnw94/24JHDYQbBQAJ

2.6 Preprocessing image on server side

Most imaging systems need to perform complex CPU-intensive image processing functions on data. These computations tax the software a great deal and often result in the decrease of overall software performance. As a remedy, a client/server model is useful in distributing the work. Multiple clients can connect to the same server, making each of the clients smaller in code size. If the server is running on a much faster platform than the clients', each client will attain most of the image processing speed of the server to which it is connected. Hence, most of the image preprocessing operations have been executed on the server. Considering the image preprocessing needs on server, flask framework(a python framework) with Nginx web server and Gunicorn python WSGI http server were finalized for implementation after comparing with other popular options like Apache and mod_wsgi.

2.7 Comparison of Different options for implementation – (Native apps and hybrid apps)

Various alternatives like Hybrid applications and React Native can also provide application build features. Their comparison was done with native application development based on several aspects as discussed below:

Native vs Hybrid

When it comes to app development there are two major methods of building an app: Native and Hybrid. Mobile devices use various Operating Systems but the most popular include Android and iOS. Both the OSs have different architecture and hence use different tools and programming languages for app development. They have also provided their own set of tools that can be used to leverage full system potential while developing apps for their platform. Using the system specific tools to develop platform specific apps is called the Native method of development.

Hybrid method involves bundling a common web application and running them on different platforms. Android, iOS and other platforms provide support for this method of development wherein a website is essentially run as an app on the OS, this method is called Hybrid development.

Although Hybrid development sounds better since it reduces efforts, Native app development has several advantages. We have discussed some major advantages below :

1. Full integration of the device :

Since Native development uses tools specified by the platform developers it is possible to leverage full system hardware as well as software potential. Since Native apps run essentially on browsers they are very limited on the system resources they can use, for example camera feeds, GPS, calendar and microphone. Since our app requires extensive use of camera, Native development is a better choice.

2. Work without internet connection :

Native apps do not require an internet connection to work as opposed to hybrid apps for which internet support is a must. Offline mode is a major part of our app and hence Native development is a better choice.

3. Performance :

Native apps excel in performance as compared to the hybrid apps. Even though browsers nowadays are fast, the hybrid apps still have a lot of processing in form of translation of web functions to native elements. Hybrid apps thus have a middle layer they must go through. This can affect various features like animation, app response time, etc. An OCR engine is already computationally heavy and using Hybrid development would degrade the performance a lot. So Native development is better.

4. Look and feel :

Native applications make use of system capabilities and functions to provide a rich feature set as well as a fluid responsive and eye catchy GUI. Hybrid apps are restricted to the use of web components as opposed to native Views. So Native development is better.

All the points discussed above are sufficient to conclude that Native development is a better choice for the application at hand.

Chapter 3

Problem Statement

Accurate and fast OCR Solution on Android for Devanagari Script which can work both online and offline, and is user friendly

System Requirement Specification

3.1 Functional Requirements

1. Scan document using phone camera
2. Import image from local storage
3. Resize and select a part of the image
4. Select language for OCR
5. Generate text from the image by running OCR procedures on the image
6. Convert the text file into other file formats
7. Save the file to local storage

3.2 System Requirements

1. At least 2 GB of RAM required for operating Tesseract OCR engine
2. More than 5 MP camera to get a clean picture

3.3 Software Requirements

1. Tesseract 4.1.0 - OCR engine required for executing the OCR process on input images.
2. Leptonica 1.78.0 - Support library for Tesseract used for image processing and image analysis.

3. libjpeg v9c - Support library for Tesseract used for manipulation of jpeg images.
4. libpng 1.6.37 - Support library for Tesseract used for manipulation of png images.
5. Android 4.1 (API 16) or higher - Supporting OS.
6. v4.0.0 trained data files for Tesseract - Trained data files required for Tesseract during the OCR process.
7. PDF Renderer - Module to convert text output files to PDF format.
8. NumPy - Python library that provides support for advanced mathematical functions.
9. Open CV - Library used for image processing.
10. Flask - Python web microframework used for developing web applications.
11. Nginx - Web server
12. Gunicorn - Python web server Gateway interface HTTP server.
13. Volley - HTTP library used for networking in Android apps.
14. Tesseract4Android - Wrapper for the Tesseract 4.1.0 to be used in Android.

3.4 Non Functional Requirements

1. More than 90% accuracy of generated text
2. Optimal processing time
3. Should not overload the device

Chapter 4

System Design & Architecture

4.1 Components in the application

The design for the application is of the utmost importance. The application has to do the OCR processing at the client side itself. This creates extra complexity for the system, because the resources required for the OCR engine to work and whether the device supports these requirements should be taken into account.

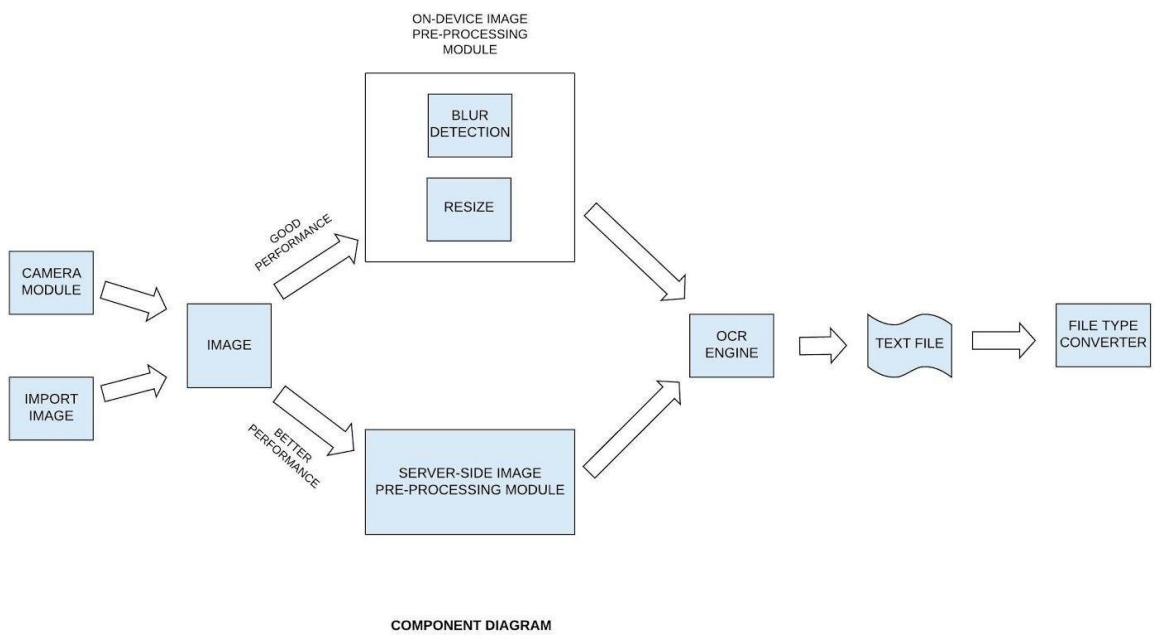


Figure 4: Component Diagram

Discussing the diagram:

An image is provided to the application either by the inbuilt camera module or specifically imported by the user. The image can be uploaded to the server for pre-processing phase to improve the image quality which ensures accuracy in the OCR engine. OCR engine extracts text from the image and puts it into a text file which can be converted to some other file formats as well.

4.2 Server side components

The server-side components are illustrated using the diagram below:

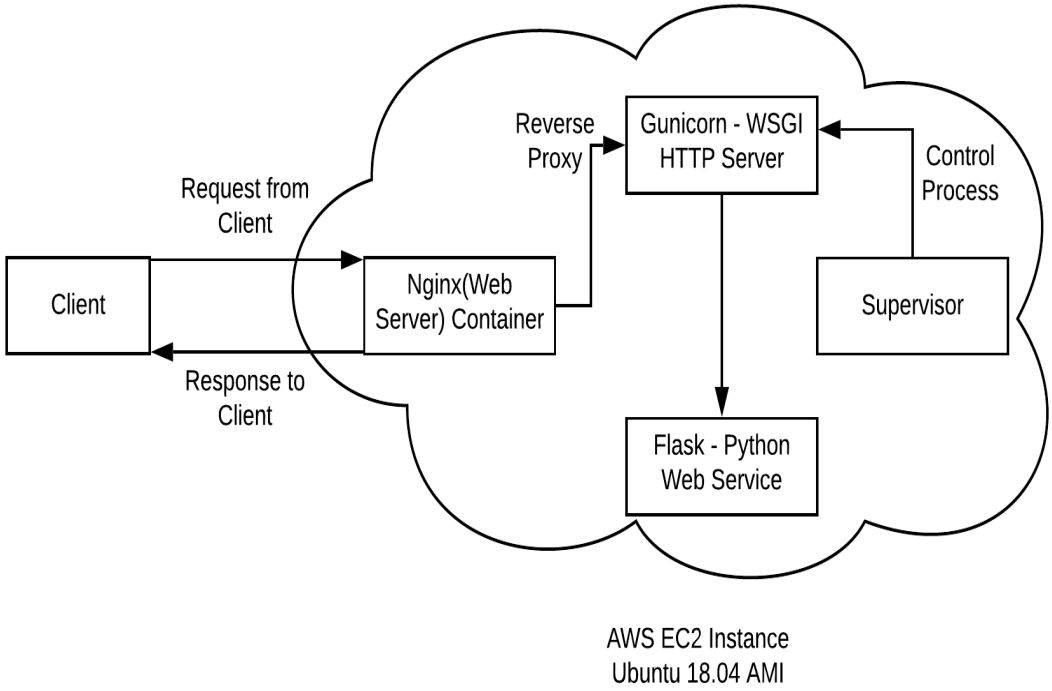


Figure 5: Server-side components

4.3 Image preprocessing functionalities

The following diagram shows the sequential flow of image preprocessing operations :

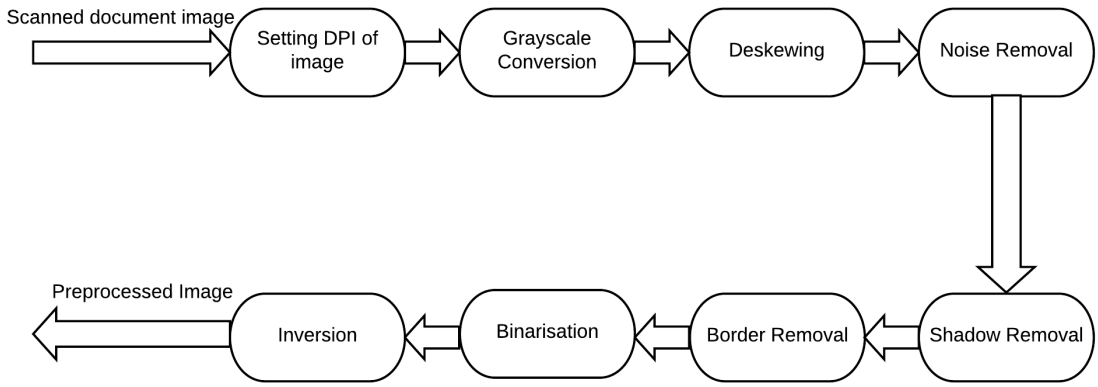


Figure 6: Image Preprocessing process flow diagram

4.4 Use case diagram

The interaction of the user with the system is shown in the following diagram:

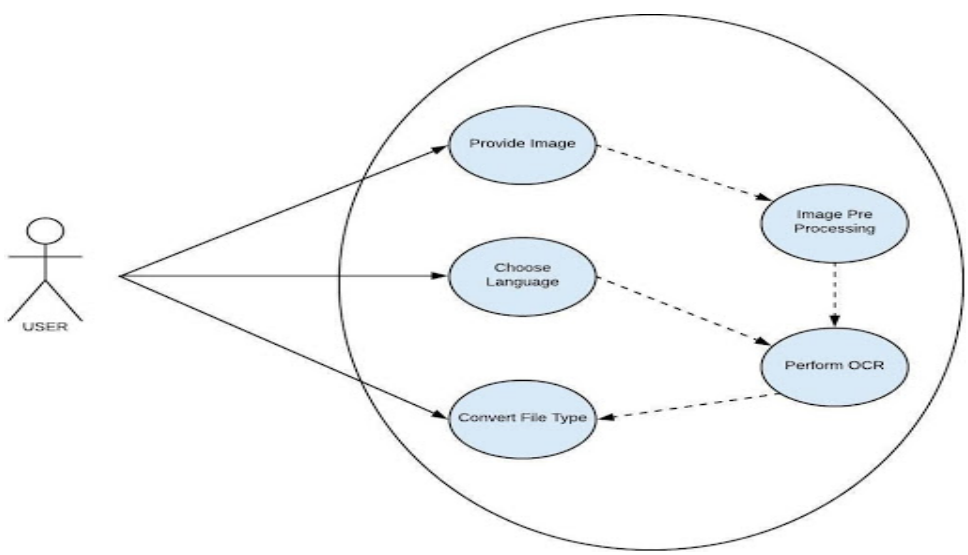


Figure 7: Use case diagram

Chapter 5

Implementation

5.1 Client side

5.1.1 Client side implementation

The application is developed in Android Studio. The app requires permissions for reading and writing to external storage, internet permissions and permission to access network state. The feature to download the models for the required language from the tesseract github repository is added in the application. For developing the application, Java and Kotlin languages are used. Also various C++ static and shared libraries are used in the project e. g. Leptonica, Tesseract, libjpeg, libpng, etc. These C++ libraries are compiled into native libraries that Gradle can package with the APK. The Android.mk file is used in this process. This is like a makefile for the ndk-build command that builds the wrapper. The libraries are included in the final build by mentioning them in the application wide Application.mk makefile. The Java or Kotlin code can then call functions in the native library through the JNI. Specifically Tesseract4Android is used which provides a Java API for accessing natively-compiled Tesseract and Leptonica APIs. This Tesseract4Android module is downloaded from github. This module provides API for many tasks such as initializing the OCR engine, setting the language for OCR, setting the OCR Engine mode(there are four OCR engine modes in Tesseract 4 which can be chosen based on accuracy and speed requirements), setting the page segmentation mode(there are 14 different page segmentation modes in Tesseract 4 of which PSM_SPARSE_TEXT and PSM_SINGLE_BLOCK are selected for generalized use) and finally carrying out the OCR. The functions in this module are called from the application logic. Volley networking library is used to send http requests to the local server. The client-server architecture is based on the REST model. The Android Network Profiler was used to capture the HTTP packets and thus debug the problems in client server interaction.

5.1.2 User Interface of the application

The following snapshots show the User Interface of this application :

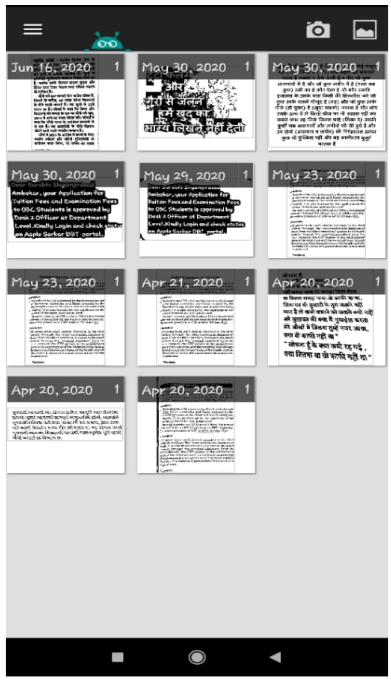


Figure 8: Launcher Activity - Grid of documents

Figure 9: Selecting image from Gallery

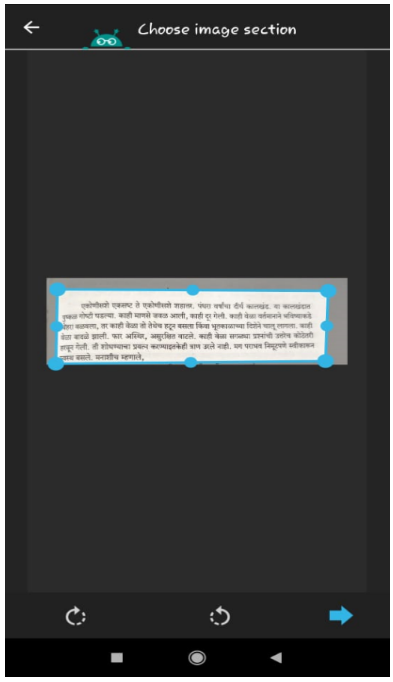


Figure 10: Cropping the image to choose the text section

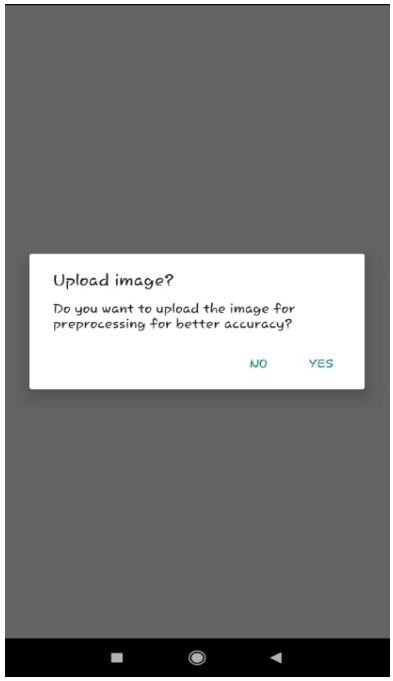


Figure 11: Dialog box asking user if they want to upload the image

Figure 12: 'Loading Spinner' while uploading the image

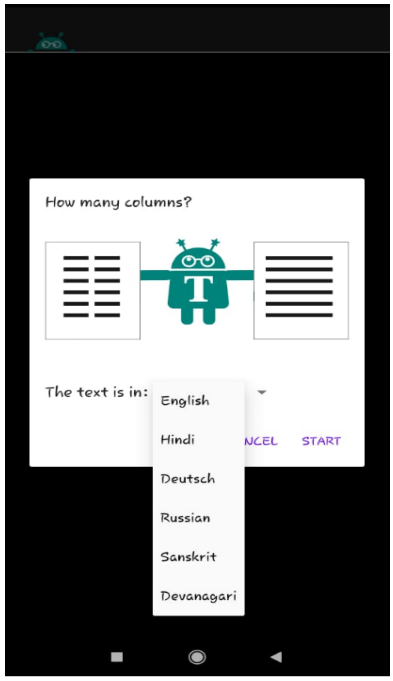


Figure 13: Selecting the language/script of the text

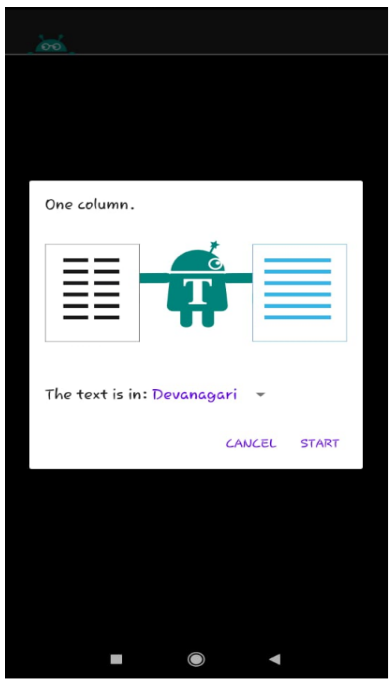


Figure 14: Selecting the layout between Simple or Complex

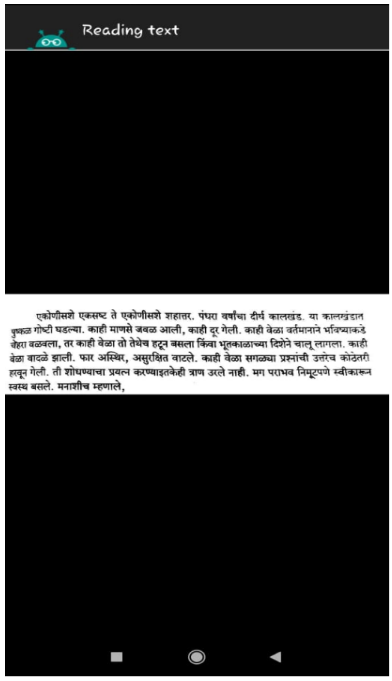


Figure 15: Activity displayed while performing the OCR on the preprocessed image

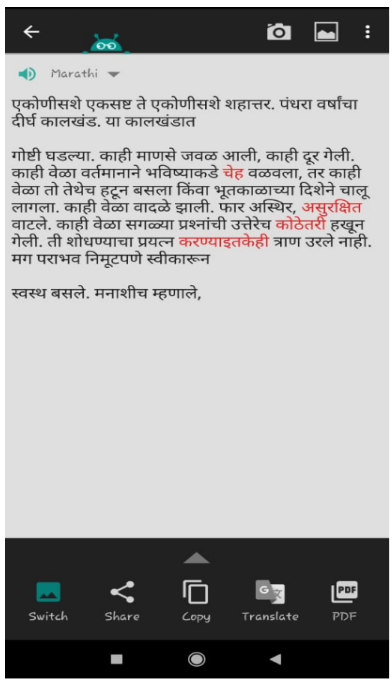


Figure 16: Final result of OCR

5.2 Server side

5.2.1 Server side implementation

Server architecture is hosted on Amazon Web Services(AWS) EC2(Elastic Compute Cloud) instance. AMI(Amazon Machine Image) for Ubuntu 18.04 is added in the EC2 instance. The flask framework is used to execute the image processing codes using opencv, imagemagick, pillow, numpy, etc. As flask framework lacks proper web server functionality, Nginx web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache is used. A traditional web server does not understand or have any way to run Python applications. A Web Server Gateway Interface (WSGI) server implements the web server side of the WSGI interface for running Python web applications. So, Gunicorn is used as Web Server Gateway Interface. The image processing pipeline is implemented after reading the recommendations given on Tesseract github repository and studying the similar work by other people using blogs, forums and github repositories. The image preprocessing web service was first tested using curl and postman commands and then integrated with Android application.

5.2.2 Snapshots of server side

The following screenshots show the configuration of server side architecture :

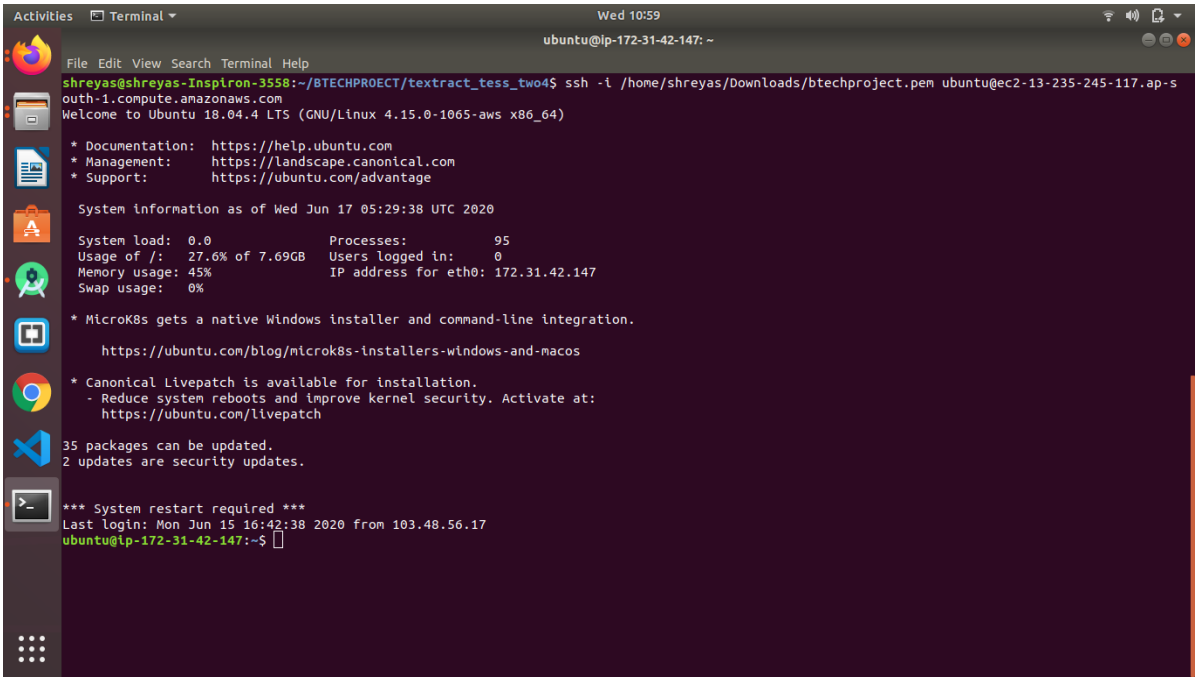


Figure 17: Connecting to the AWS EC2 instance with the help of SSH

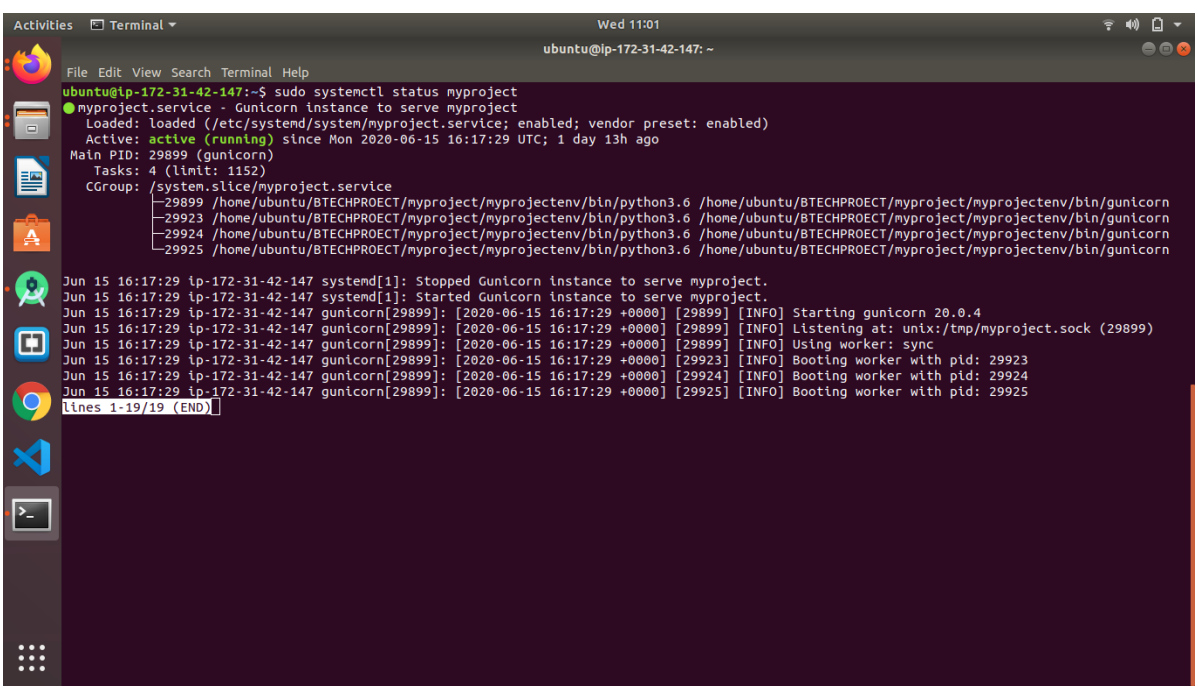


Figure 18: Status of Gunicorn WSGI

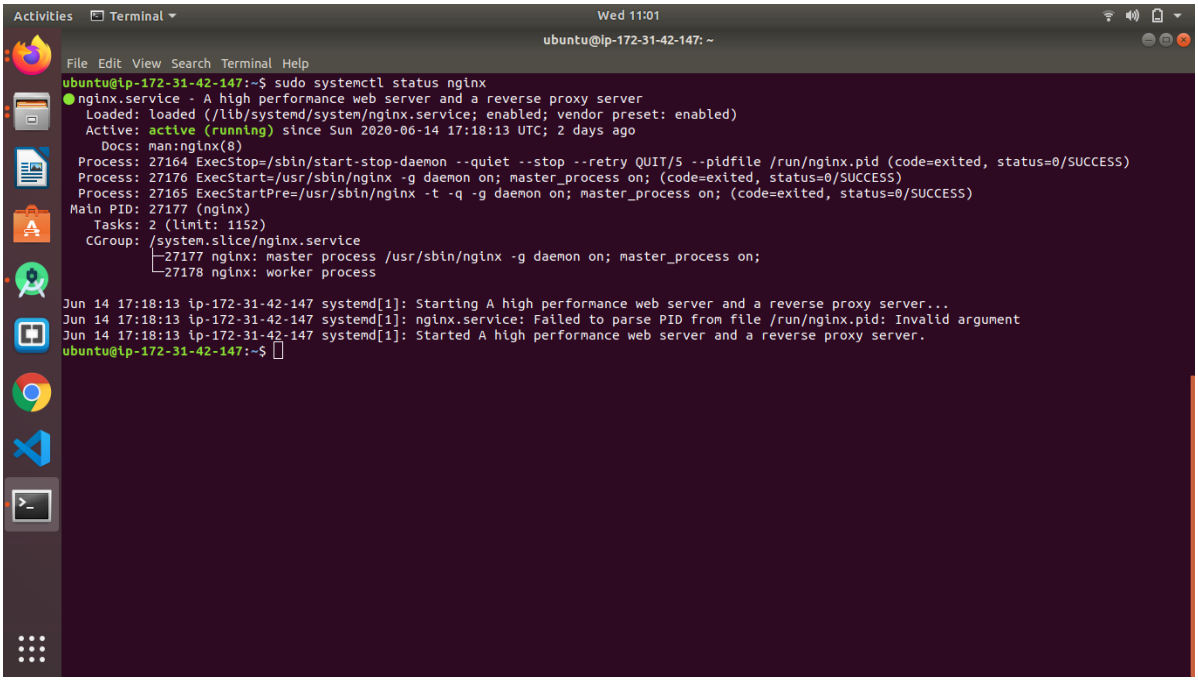


Figure 19: Status of Nginx Web Server

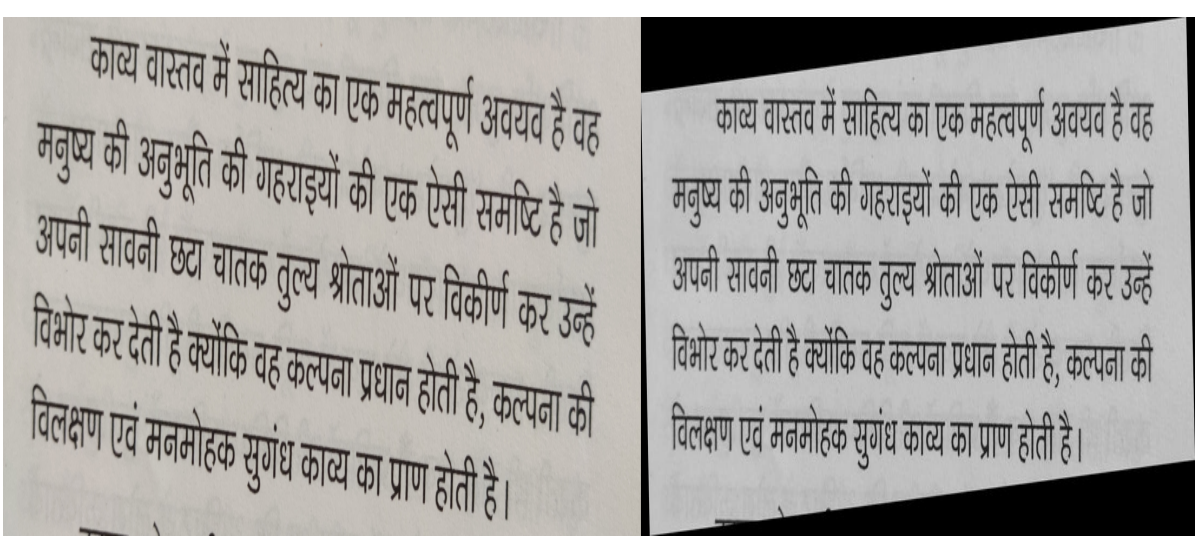
5.2.3 Image preprocessing operations

The following are the image preprocessing operations performed on the server side along with the sample output images for each operation :

Image Rescaling - Tesseract 4.0 works best on images with a DPI of atleast 300. DPI lower than 200 produces unclear and incomprehensible results, whereas DPI above 600 increases size of the output image without significant gains in image quality. The image is set to RGB mode with DPI of 300 to save it in JPEG format.

Grayscale Conversion - The pixel values of the grayscaled image ranges from 0 to 255. The upside of doing this operation is that detection of edges become easier, model complexity reduces and OCR processing time reduces. cv2.cvtColor() method in cv2(a python interface for OpenCV) is used to convert the image from RGB color space to grayscale color space.

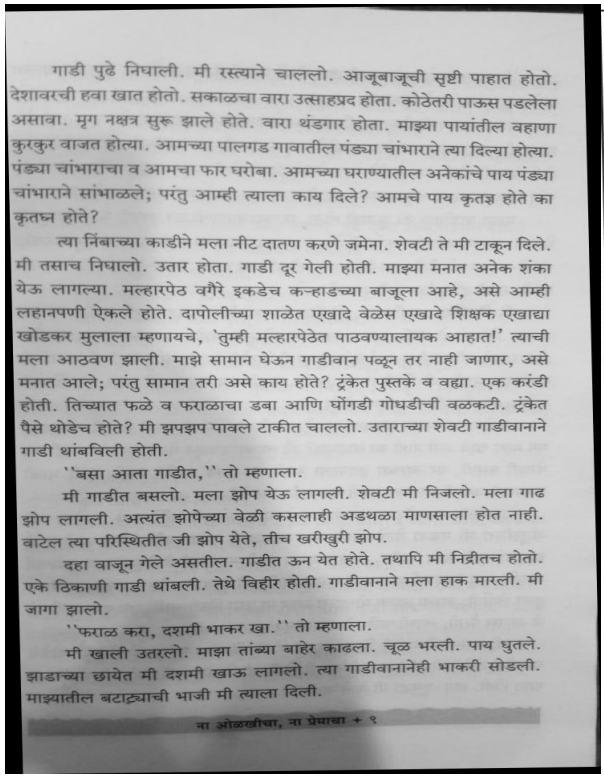
Deskewing - This operation rotates the input image by the amount of skew detected. It initially finds all the pixel values in the foreground text and constructs a rotated bounded box around this text region. Following this, an angle value is returned in the range of [-90, 0) degrees. As the image rotates in the clockwise direction, the angle increases from -90 to 0. When 0 is reached, the angle is reset to -90 degrees. Finally, an affine transformation is applied in order to correct the skew. cv2.minAreaRect() method is used which returns the minimum area rectangular box bounding the text region along with the angle of rotation. cv2.getRotationMatrix2D() method returns the rotation matrix used by cv2.warpAffine() to produce the deskewed image.



Shadow Removal - It involves two operations of dilation and median blurring. In dilation, a structuring element(a matrix of pixel values) is selected and moved over the image such that the anchor point(centre of structuring element) coincides with pixels in the image. The maximum of all the pixels in the structuring element is computed at every pixel in the image and the anchor point is replaced by this maximum value. This operation dilates or widens the text region in the image. A kernel of all 1s with a size of 7*7 is used. All elements in the kernel are chosen to be 1s so that only the region around the text would get brightened. The pixel values in the other regions of the image would remain unchanged. The kernel size of 7*7 is chosen as bigger kernel size increases processing time and a smaller kernel size doesn't dilate the text region properly. cv2.dilate() method accomplishes this task.

Noise removal is performed using Median blurring to remove salt and pepper noise(few pixels in the image are extremely noisy). It replaces the pixel value in the image by calculating the median of neighbouring pixel values. cv2.medianBlur() method is used to perform median blurring.

Thus, dilation aids in getting rid of the text, and median blur further increases the blur in text. The resulting image is an image with shadow. In order to get rid of this shadow, an absolute difference is calculated between the original image and the image just obtained. The resultant image has white text on black background. As black text on white background is preferred, the image is inverted.



गाडी पुढे निघाली. मी रस्त्याने चाललो. आजूबाजूची सृष्टी पाहात होतो. देशावरची हवा खात होतो. सकाळचा वारा उत्साहप्रद होता. कोठेतरी पाऊस पडलेला असावा. मृग नक्षत्र सुरू झाले होते. वारा थंडगार होता. माझ्या पायांतील वहाणा कुरकुर वाजत होत्या. आमच्या पालगड गावातील पंड्या चांभाराने त्या दिल्या होत्या. पंड्या चांभाराचा व आमचा फार घरोबा. आमच्या घराण्यातील अनेकांचे पाय पंड्या चांभाराने सांभाळले; परंतु आम्ही त्याला काय दिले? आमचे पाय कृतज्ञ होते का कृतघ्न होते?

त्या निंबाच्या काडीने मला नीट दातण करणे जमेना. शेवटी ते मी टाकून दिले. मी तसाच निघालो. उतार होता. गाडी दूर गेली होती. माझ्या मनात अनेक शंका येऊ लागल्या. मल्हारपेठ वगैरे इकडेच कऱ्हाडच्या बाजूला आहे, असे आम्ही लहानपणी ऐकले होते. दापोलीच्या शाळेत एखादे वेळेस एखादे शिक्षक एखाद्या खोडकर मुलाला म्हणायचे, 'तुम्ही मल्हारपेठेत पाठवण्यालायक आहात!' त्याची मला आठवण झाली. माझे सामान घेऊन गाडीवान पळून तर नाही जाणार, असे मनात आले; परंतु सामान तरी असे काय होते? ट्रंकेत पुस्तके व वह्या. एक करंडी होती. तिच्यात फळे व फराळाचा डबा आणि घोंगडी गोधडीची वळकटी. ट्रंकेत पैसे थोडेच होते? मी झपझप पावले टाकीत चाललो. उताराच्या शेवटी गाडीवानाने गाडी थांबविली होती.

''बसा आता गाडीत,'' तो म्हणाला.

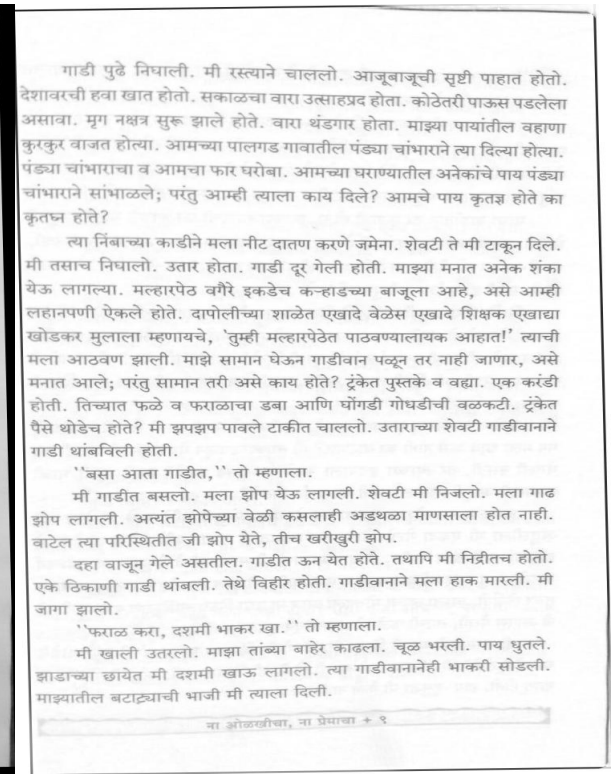
मी गाडीत बसलो. मला झोप येऊ लागली. शेवटी मी निजलो. मला गाढ झोप लागली. अत्यंत झोपेच्या वेळी कसलाही अडथळा माणसाला होत नाही. वाटेल त्या परिस्थितीत जी झोप येते, तीच खरीखुरी झोप.

दहा वाजून गेले असतील. गाडीत ऊन येत होते. तथापि मी निद्रीतच होतो. एके ठिकाणी गाडी थांबली. तेथे विहीर होती. गाडीवानाने मला हाक मारली. मी जागा झालो.

''फराळ करा, दशमी भाकर खा.'' तो म्हणाला.

मी खाली उतरलो. माझा तांब्या बाहेर काढला. चूळ भरली. पाय धुतले. झाडाच्या छायेत मी दशमी खाऊ लागलो. त्या गाडीवानानेही भाकरी सोडली. माझ्यातील बटाट्याची भाजी मी त्याला दिली.

ना ओळखीचा, ना प्रेमाचा + ९



गाडी पुढे निघाली. मी रस्त्याने चाललो. आजूबाजूची सृष्टी पाहात होतो. देशावरची हवा खात होतो. सकाळचा वारा उत्साहप्रद होता. कोठेतरी पाऊस पडलेला असावा. मृग नक्षत्र सुरू झाले होते. वारा थंडगार होता. माझ्या पायांतील वहाणा कुरकुर वाजत होत्या. आमच्या पालगड गावातील पंड्या चांभाराने त्या दिल्या होत्या. पंड्या चांभाराचा व आमचा फार घरोबा. आमच्या घराण्यातील अनेकांचे पाय पंड्या चांभाराने सांभाळले; परंतु आम्ही त्याला काय दिले? आमचे पाय कृतज्ञ होते का कृतघ्न होते?

त्या निंबाच्या काडीने मला नीट दातण करणे जमेना. शेवटी ते मी टाकून दिले. मी तसाच निघालो. उतार होता. गाडी दूर गेली होती. माझ्या मनात अनेक शंका येऊ लागल्या. मल्हारपेठ वगैरे इकडेच कऱ्हाडच्या बाजूला आहे, असे आम्ही लहानपणी ऐकले होते. दापोलीच्या शाळेत एखादे वेळेस एखादे शिक्षक एखाद्या खोडकर मुलाला म्हणायचे, 'तुम्ही मल्हारपेठेत पाठवण्यालायक आहात!' त्याची मला आठवण झाली. माझे सामान घेऊन गाडीवान पळून तर नाही जाणार, असे मनात आले; परंतु सामान तरी असे काय होते? ट्रंकेत पुस्तके व वह्या. एक करंडी होती. तिच्यात फळे व फराळाचा डबा आणि घोंगडी गोधडीची वळकटी. ट्रंकेत पैसे थोडेच होते? मी झपझप पावले टाकीत चाललो. उताराच्या शेवटी गाडीवानाने गाडी थांबविली होती.

''बसा आता गाडीत,'' तो म्हणाला.

मी गाडीत बसलो. मला झोप येऊ लागली. शेवटी मी निजलो. मला गाढ झोप लागली. अत्यंत झोपेच्या वेळी कसलाही अडथळा माणसाला होत नाही. वाटेल त्या परिस्थितीत जी झोप येते, तीच खरीखुरी झोप.

दहा वाजून गेले असतील. गाडीत ऊन येत होते. तथापि मी निद्रीतच होतो. एके ठिकाणी गाडी थांबली. तेथे विहीर होती. गाडीवानाने मला हाक मारली. मी जागा झालो.

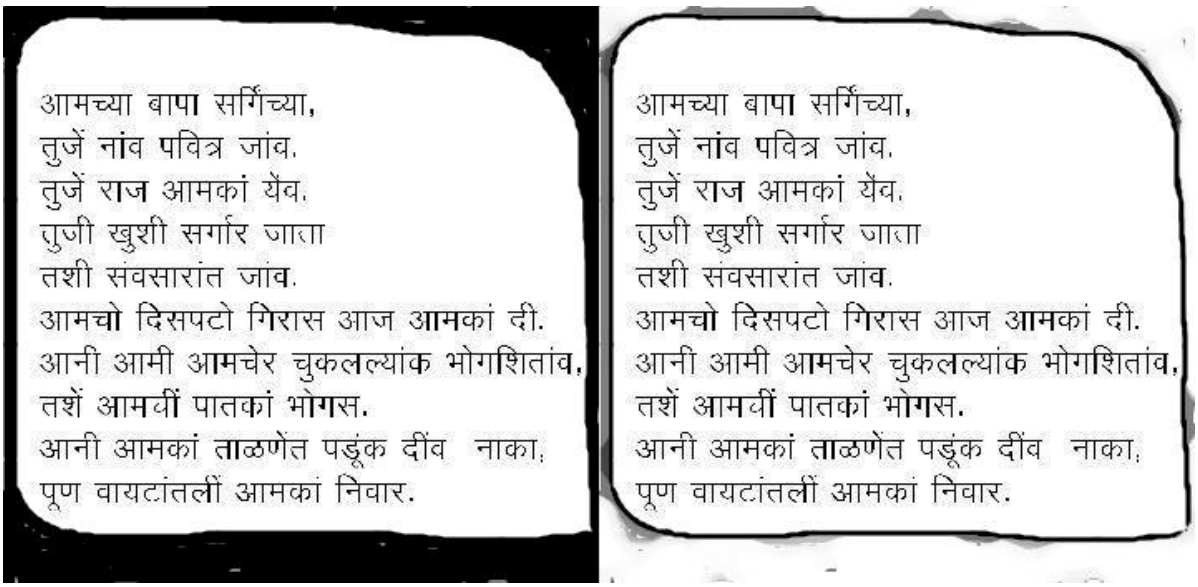
''फराळ करा, दशमी भाकर खा.'' तो म्हणाला.

मी खाली उतरलो. माझा तांब्या बाहेर काढला. चूळ भरली. पाय धुतले. झाडाच्या छायेत मी दशमी खाऊ लागलो. त्या गाडीवानानेही भाकरी सोडली. माझ्यातील बटाट्याची भाजी मी त्याला दिली.

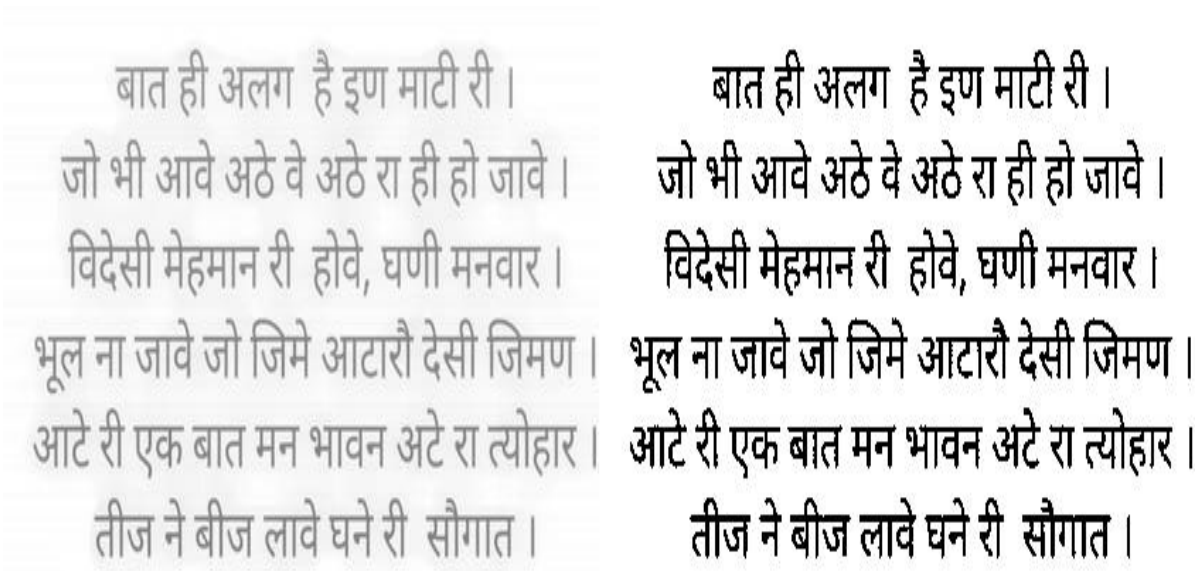
ना ओळखीचा, ना प्रेमाचा + ९

Border Removal - In this functionality, a mask of all 0 bits is applied to the entire image. Furthermore, a contour(a set of continuous points along a boundary having same color intensity) is drawn on this mask selecting only the regions of interest. This contour of black region is inverted and bitwise-or operation is performed between the original image and image just obtained to get borderless image.

cv2.findcontours() method is used to find the contours in an image. It returns a python list of all the contours in an image. Each individual contour is a numpy array of (x, y) co-ordinates. Sometimes, contours are present inside contours. As only the outermost contour is required for border removal, contour retrieval mode is set to RETR_EXTERNAL which returns only the outermost contours. Besides, to remove redundant points returned by findcontours(), contour approximation method is set to CHAIN_APPORX_SIMPLE. This approximation method helps in saving memory.

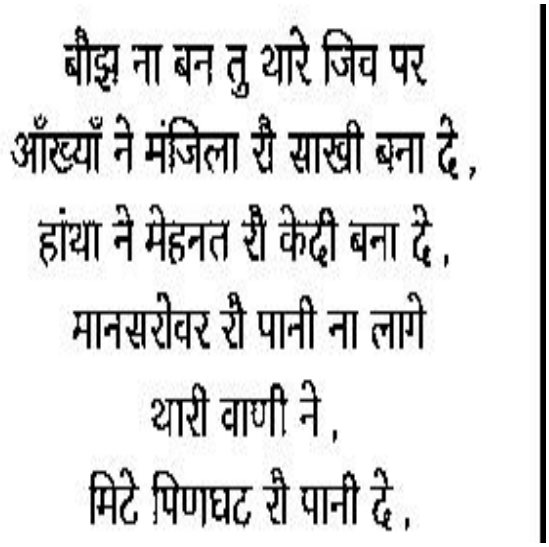


Binarisation - This operation converts the image to black and white. Otsu's thresholding is used to accomplish this. Otsu's thresholding works by classifying pixels into two classes - foreground and background such that within class variance is minimal. The pixel value for which the within class variance is minimal is selected as the threshold value. cv2.threshold() method with cv2.THRESH_BINARY + cv2.THRESH_OTSU parameter performs the desired operation.



Inversion - As Tesseract performs better on black text on white background, inver-

sion has been done.



Chapter 6

Results

The android application can successfully carry out on device OCR using the wrapper library of Tesseract 4 with acceptable speed. The option to preprocess an image for better accuracy has been given in which the image will be sent to the server for preprocessing via the image upload module. The flask framework for running the image processing scripts has been setup along with Nginx and Gunicorn on the server side. The server side architecture has been hosted on AWS EC2 instance. The accuracy of this application has been calculated using the word error rate metric. Testing has been carried out on more than 500 images. Different variations like shadow, border, skewness, background color, font type, font style(bold and italic), image quality and folds have been introduced in the images.

The average word error rate per image and total word error rate of some languages using Devanagari script on which the OCR has been performed are as follows:

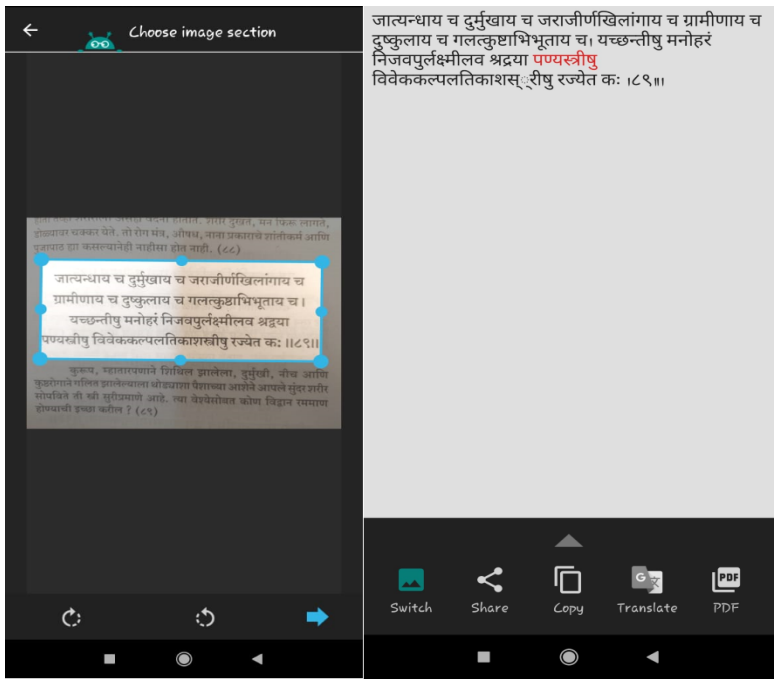
Language	Average word error rate(%)	Total word error rate(%)
Hindi	20.25	13.42
Marathi	10.04	5.75
Rajasthani	17.9	17.28
Nepalese	14.53	11.75
Konkani	11.03	10.92
Sanskrit	38.24	37.82

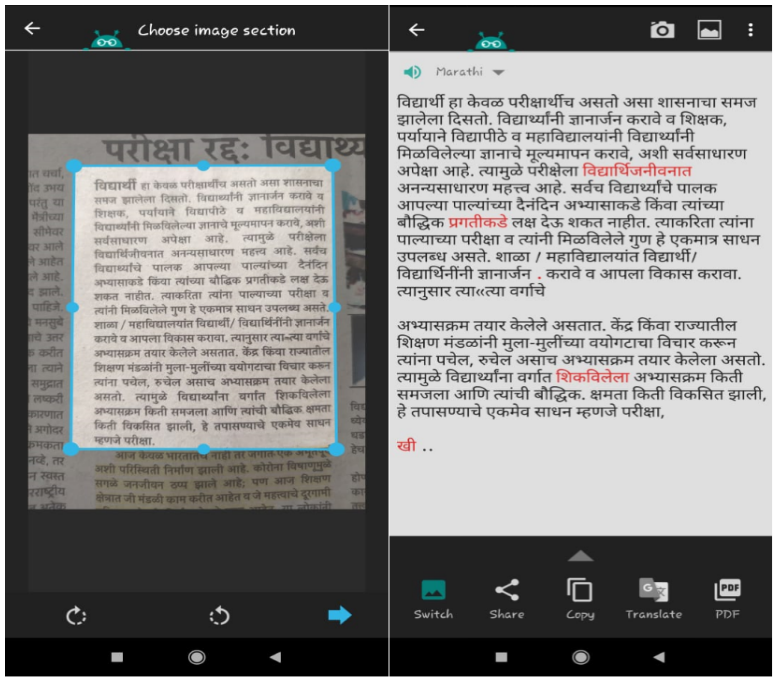
Table 3: Error rates

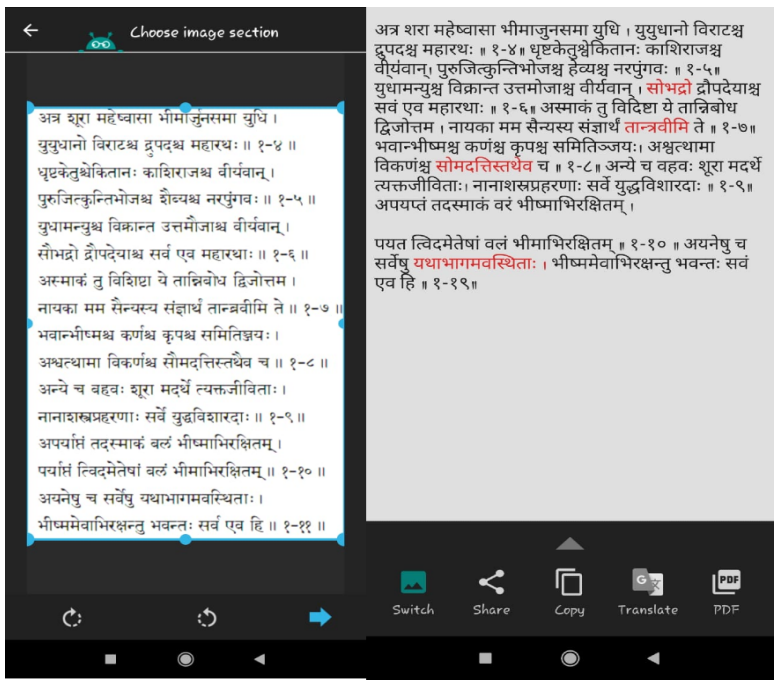
According to the accuracy results shown in Table 2, the word error rates for Hindi, Marathi and Nepalese using the LSTM based Tesseract(without image preprocessing and post-processing) are 30.19%, 32.79% and 42.39% respectively. The percentage change in

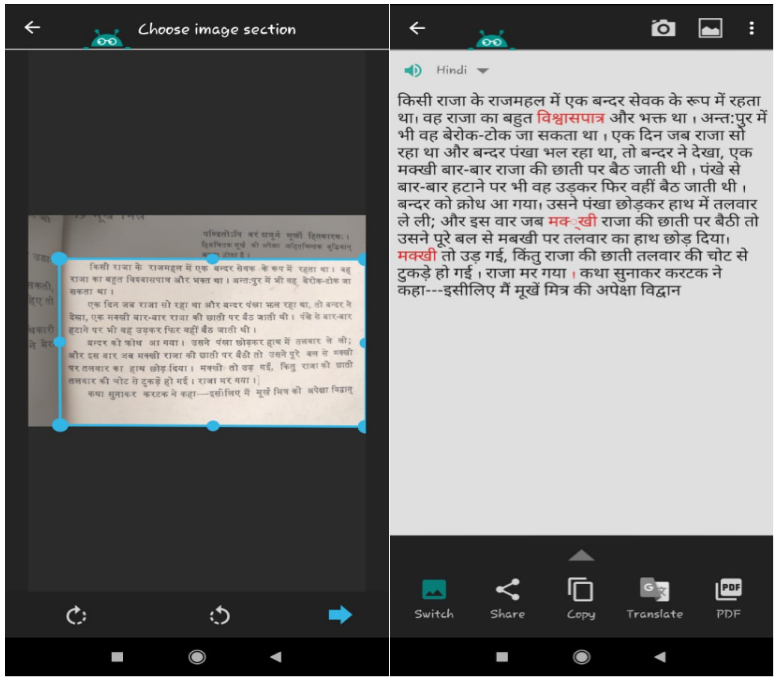
average word error rates per image observed for these languages using this application are 9.94%, 22.75% and 27.86% respectively. Thus, the overall accuracy after OCR has improved considerably compared to implementing OCR without image preprocessing.

The following examples show the original image and the snapshot of corresponding OCR output:

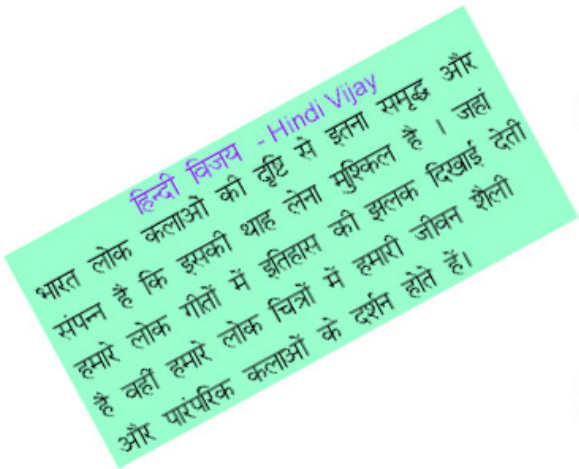


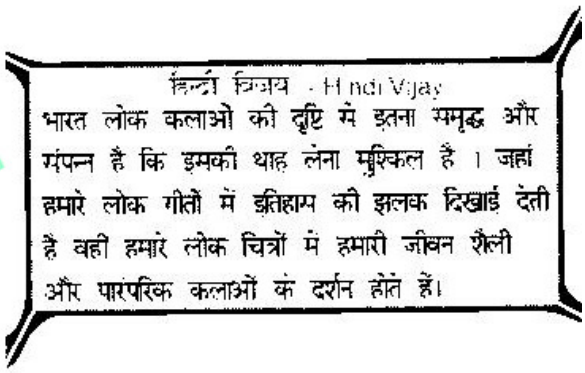

Choose image section
Marathi
विद्यार्थी हा केवळ परीक्षार्थीच असतो असा शासनाचा समज झालेला दिसतो. विद्यार्थ्यांनी ज्ञानार्जन करावे व शिक्षक, पर्यायाने विद्यापीठे व महाविद्यालयांनी मिळविलेल्या ज्ञानाचे मूल्यमापन करावे, अशी सर्वसाधारण अपेक्षा आहे. त्यामुळे परीक्षेला विद्यार्थिजनीवनात अनन्यसाधारण महत्त्व आहे. सर्वच विद्यार्थ्यांचे पालक आपल्या पाल्यांच्या दैनंदिन अभ्यासाकडे किंवा त्यांच्या बौद्धिक प्रगतीकडे लक्ष देऊ शकत नाहीत. त्याकरिता त्यांना पाल्याच्या परीक्षा व त्यांनी मिळविलेले गुण हे एकमात्र साधन उपलब्ध असते. शाळा / महाविद्यालयांत विद्यार्थी/ विद्यार्थिनींनी ज्ञानार्जन . करावे व आपला विकास करावा. त्यानुसार त्या«त्या वर्गाचे
अभ्यासक्रम तयार केलेले असतात. केंद्र किंवा राज्यातील शिक्षण मंडळांनी मुला-मुलींच्या वयोगटाचा विचार करून त्यांना पचेल, रुचेल असाच अभ्यासक्रम तयार केलेला असतो. त्यामुळे विद्यार्थ्यांना वर्गात शिकविलेला अभ्यासक्रम किती समजला आणि त्यांची बौद्धिक. क्षमता किती विकसित झाली, हे तपासण्याचे एकमेव साधन म्हणजे परीक्षा,
खी ..
Switch
Share
Copy
Translate
PDF


Choose image section
अत्र शूरा महेष्वासा भीमार्जुनसमा युधि ।
युयुधानो विराटश्च द्रुपदश्च महारथः ॥ १-४ ॥
धृष्टकेतुश्चेकितानः काशिराजश्च वीर्यवान् ।
पुरुजित्कुन्तिभोजश्च शैब्यश्च नरपुंगवः ॥ १-५ ॥
युधामन्युश्च विक्रान्त उत्तमौजाश्च वीर्यवान् ।
सौभद्रो द्रौपदेयाश्च सर्व एव महारथाः ॥ १-६ ॥
अस्माकं तु विशिष्टा ये तान्निबोध द्विजोत्तम ।
नायका मम सैन्यस्य संज्ञार्थं तान्ब्रवीमि ते ॥ १-७ ॥
भवान्भीष्मश्च कर्णश्च कृपश्च समितिञ्जयः ।
अश्वत्थामा विकर्णश्च सौमदत्तिस्तथैव च ॥ १-८ ॥
अन्ये च बहवः शूरा मदर्थे त्यक्तजीविताः ।
नानाशस्त्रप्रहरणाः सर्वे युद्धविशारदाः ॥ १-९ ॥
अपर्याप्तं तदस्माकं बलं भीष्माभिरक्षितम् ।
पर्याप्तं त्विदमेतेषां बलं भीमाभिरक्षितम् ॥ १-१० ॥
अयनेषु च सर्वेषु यथाभागमवस्थिताः ।
भीष्ममेवाभिरक्षन्तु भवन्तः सर्व एव हि ॥ १-११ ॥
अत्र शरा महेष्वासा भीमाजुनसमा युधि । युयुधानो विराटश्च द्रुपदश्च महारथः ॥ १-४॥ धृष्टकेतुश्चेकितानः काशिराजश्च वीर्यवान्। पुरुजित्कुन्तिभोजश्च हेव्यश्च नरपुंगवः ॥ १-५॥ युधामन्युश्च विक्रान्त उत्तमोजाश्च वीर्यवान् । सोभद्रो द्रौपदेयाश्च सर्व एव महारथाः ॥ १-६॥ अस्माकं तु विदिष्टा ये तान्निबोध द्विजोत्तम । नायका मम सैन्यस्य संज्ञार्थं तान्त्रवीमि ते ॥ १-७॥ भवान्भीष्मश्च कर्णश्च कृपश्च समितिञ्जयः। अश्वत्थामा विकर्णश्च सोमदत्तिस्तथैव च ॥ १-८॥ अन्ये च वहवः शूरा मदर्थे त्यक्तजीविताः। नानाशस्त्रप्रहरणाः सर्वे युद्धविशारदाः ॥ १-९॥ अपयप्तं तदस्माकं वरं भीष्माभिरक्षितम् ।
पयत त्विदमेतेषां वलं भीमाभिरक्षितम् ॥ १-१० ॥ अयनेषु च सर्वेषु यथाभागमवस्थिताः । भीष्ममेवाभिरक्षन्तु भवन्तः सवं एव हि ॥ १-१९॥
Switch
Share
Copy
Translate
PDF



The following examples show the original image and the image processed using scripts executed on the server.



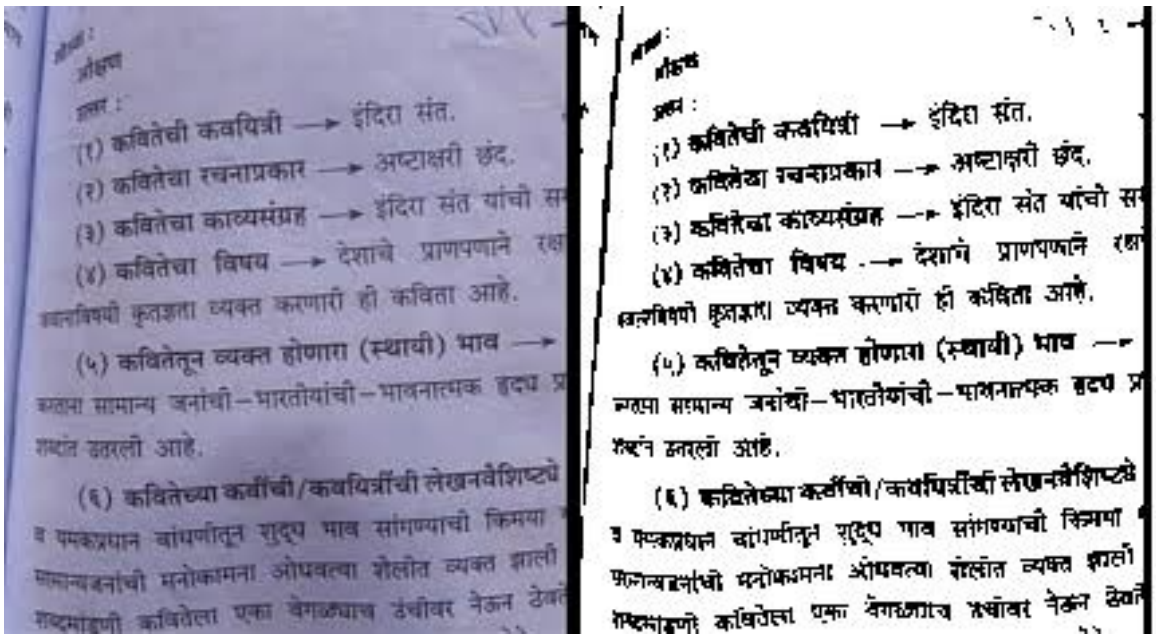


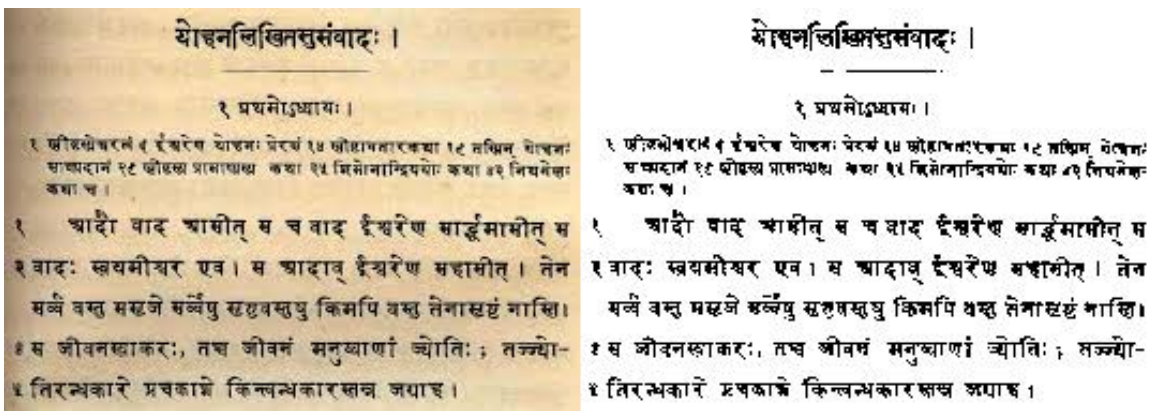
सादा पानामाथि एउटा
पात राख्नुहोस् ।
सिसाकलमले पातको
वरिपरि कोर्नुहोस् ।

१०४

सादा पानामाथि एउटा
पात राख्नुहोस् ।
सिसाकलमले पातको
वरिपरि कोर्नुहोस् ।

१०४





प्रतिबंधित क्षेत्रासाठी...

शाळा व्यवस्थापन समितीच्या मुख्य शिफारसी व तज्ज्ञ व्यक्तींच्या शिफारसींनुसार प्रतिबंधित क्षेत्रातील जिल्हाधिकारी-महापालिका आयुक्तांनी शाळा कधी सुरू करायच्या याचा निर्णय घ्यावा.

प्रतिबंधित क्षेत्रासाठी...

शाळा व्यवस्थापन समितीच्या मुख्य शिफारसी व तज्ज्ञ व्यक्तींच्या शिफारसींनुसार प्रतिबंधित क्षेत्रातील जिल्हाधिकारी-महापालिका आयुक्तांनी शाळा कधी सुरू करायच्या याचा निर्णय घ्यावा.

॥ अथ ध्यानम् ॥

हस्ति इंद्र आननं इंद्र चूडं अरुणच्छायं त्रिनेत्रं रसात्
आश्लिष्टं प्रियया सपद्म करया स्व अंकस्थया संततं ।
बीजापूर गदा धनु: त्रिशिखयुक् चक्र अब्ज पाश उत्पल ।
व्रीहि अग्रस्व विषाण रत्नकलशान् हस्तै:वहन्तं भजे ॥१॥
गंडपाली गलद्दान पूर लालस मानसान् ॥

॥ अथ ध्यानम् ॥

हस्ति इंद्र आननं इंद्र चूडं अरुणच्छायं त्रिनेत्रं रसात्
आश्लिष्टं प्रियया सपद्म करया स्व अंकस्थया संततं ।
बीजापूर गदा धनु: त्रिशिखयुक् चक्र अब्ज पाश उत्पल ।
व्रीहि अग्रस्व विषाण रत्नकलशान् हस्तै:वहन्तं भजे ॥१॥
गंडपाली गलद्दान पूर लालस मानसान् ॥

स्वर्गाहुनही प्रिय अम्हांला अमुचा सुंदर भारत देश
आम्ही सारे एक, जरीही नाना जाती, नाना वेष

या भूमीच्या आम्ही कन्या, कोमल भाव मनी
फूलकोवळ्या तरी प्रसंगी होऊ रणरागिणी
आवेशाने घुसू संगरी चढवुनिया रणवेष

श्रीरामाचे, श्रीकृष्णाचे अर्जुन आहे स्मरण मनास
वीर शिवाजी, प्रताप, बाजी.. थोर आमुचा हा इतिहास
रक्तामधुनी बीज वाहते उरात भरतो नव आवेश

हिमालयापरि शीतल आम्ही, आग पेटती परि उरात
पाऊल परके पडता येथे बळ वज्राचे याच करांत
या देशाची गौरवगाथा हाच अम्हाला दे आदेश

स्वर्गाहुनही प्रिय अम्हांला अमुचा सुंदर भारत देश
आम्ही सारे एक, जरीही नाना जाती, नाना वेष

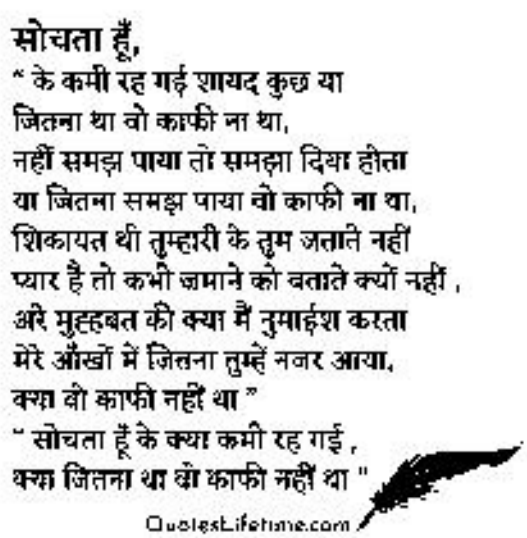
या भूमीच्या आम्ही कन्या, कोमल भाव मनी
फूलकोवळ्या तरी प्रसंगी होऊ रणरागिणी
आवेशाने घुसू संगरी चढवुनिया रणवेष

श्रीरामाचे, श्रीकृष्णाचे अर्जुन आहे स्मरण मनास
वीर शिवाजी, प्रताप, बाजी.. थोर आमुचा हा इतिहास
रक्तामधुनी बीज वाहते उरात भरतो नव आवेश

हिमालयापरि शीतल आम्ही, आग पेटती परि उरात
पाऊल परके पडता येथे बळ वज्राचे याच करांत
या देशाची गौरवगाथा हाच अम्हाला दे आदेश

तिमी को हौ बालक ? लक्ष्मीप्रसाद ।
कसका छोरा ? रामशरणको ॥
के गर्छौ ? पढाइ खेलाइ ।
हातमा के छ ? वाद्यवादन ॥
रमा लेख्छिन् अ आ इ ई ।
कृष्ण लेख्छन् क ख ग घ ॥
त्यो के लेख्यौ ? च छ ज झ ।
के पनि लेख्छौ ? ट ठ ड ढ ॥
तिमी के पढ्छौ ? त थ द ध ।
आज के भन्छौ ? प फ ब भ ॥
सुयश लेख्छन् य र ल व ।
सुरभि लेख्छिन् श ष स ह ॥

तिमी को हौ बालक ? लक्ष्मीप्रसाद ।
कसका छोरा ? रामशरणको ॥
के गर्छौ ? पढाइ खेलाइ ।
हातमा के छ ? वाद्यवादन ॥
रमा लेख्छिन् अ आ इ ई ।
कृष्ण लेख्छन् क ख ग घ ॥
त्यो के लेख्यौ ? च छ ज झ ।
के पनि लेख्छौ ? ट ठ ड ढ ॥
तिमी के पढ्छौ ? त थ द ध ।
आज के भन्छौ ? प फ ब भ ॥
सुयश लेख्छन् य र ल व ।
सुरभि लेख्छिन् श ष स ह ॥



समूहमा नमिल्ने वर्णमा गोलो घेरा लगाउनुहोस् :

आ	आ	ओ	आ
आ	औ	आ	आ
आ	आ	अः	आ
अ	आ	आ	आ

समूहमा नमिल्ने वर्णमा गोलो घेरा लगाउनुहोस् :

आ	आ	ओ	आ
आ	औ	आ	आ
आ	आ	अः	आ
अ	आ	आ	आ

शब्दका सुरुका मिल्ने वर्णसँग जोडा मिलाउनुहोस् :

शब्दका सुरुका मिल्ने वर्णसँग जोडा मिलाउनुहोस् :

Chapter 7

Future Scope

1. Post processing using Dictionary feature i. e. New Domain specific words can be added to the Dictionary for better accuracy.

2. In the application, we are using (LSTM only) traineddata file. A feature can be provided allowing user to select which traineddata file - best, fast or Legacy + LSTM to download and OEM will be selected accordingly.

3. Tesseract can be initialized for multiple languages if the image contains two or more lanuages.

4. Add the load balancing features in ngnix.

5. Smart usage of page segmentation mode. Currently, this application uses PSM_SINGLE_BLOCK for Complex Layout and PSM_SPARSE_TEXT for simple layout.

6. If ample training data is available, or a very unusual font is used, then tesseract can be trained from scratch. Besides, if some new characters are to be added to the existing character set without altering any part of the network, fine tuning can be done

Chapter 8

Conclusion

As digitisation of documents is taking place rapidly, and most of the government offices use records documented with languages based on Devanagari script, and with increasing number of smartphone users in India, it has become imperative to foster the accuracy improvement of existing OCR engines. This project was an endeavour at developing an user-friendly and easily accessible Android application integrated with Tesseract 4.0 with the inclusion of image preprocessing routines.

References

[1] "Devanagari dataset". [Online]. Available: https://gitlab.com/shreyasssa /devnagari-ocr/-/tree/master/Devanagari_Testing_Dataset/Devanagari%20OCR

[2] "Devanagari word error rate calculation". [Online]. Available: https://gitlab.com /shreyasssa/devnagari-ocr/-/raw/master/Devanagari%20Word%20Error%20Rate %20Calculation.ods?inline=false

[3] Image rescaling source code - "Improve Accuracy of OCR using Image Preprocessing". [Online]. Available: https://medium.com/cashify-engineering/improve-accuracy-of-ocr-using-image-preprocessing-8df29ec3a033

[4] Shadow removal source code - "How to remove shadow from scanned images using OpenCV? ". [Online]. Available: https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv

[5] Border removal source code - "OpenCV Python border removal preprocessing for OCR ". [Online]. Available: https://stackoverflow.com/questions/57858944/opencv-python-border-removal-preprocessing-for-ocr

[6] Deskewing source code - "deskew 0.10.2". [Online]. Available: https://github .com/sbrunner/deskew

[7] Patel, C., Patel, A.,& Patel, D. (2012), "Optical character recognition by open source OCR tool tesseract: A case study", International Journal of Computer Applications, 55(10) .

[8] Chen, H., Tsai, S. S., Schroth, G., Chen, D. M., Grzeszczuk, R., & Girod, B. (2011, September), "Robust text detection in natural images with edge-enhanced maximally stable extremal regions", 18th IEEE International Conference on Image Processing (pp. 2609-2612). IEEE, 2011.

[9] Lin, H.-Y., Hsu, C.-Y., "Optical character recognition with fast training neural network", IEEE International Conference on Industrial Technology (ICIT), pp. 1458–

1461, 2016.

[10] Jianfeng Wang, Xiaolin Hu, “Gated recurrent convolution neural network for OCR”, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[11] B.Indira, Muhammad Shuaib Qureshi, Mahaboob Sharief Shaik, Rashad Mahmood Saqib, MV Ramana Murthy, “Devanagari character recognition: a short review”, International Journal of Computer Applications (0975 –8887) Volume 59–No.6, December 2012.

[12] “Tutorial on client-server architecture - stanford document”. Accessed on: Feb 13, 2020. [Online]. Available: https://web.stanford.edu/class/ee368/ Android/Tutorial3 .pdf

[13] “Building an image processing pipeline with python”. Accessed on: Feb 22, 2020. [Online]. Available: https://pyvideo.org/pycon-us-2013/ building-an-image-processing-pipeline-with-python.html.

[14] “Comparison of accuracy of Tesseract before and after image processing”. Accessed on: Feb,15, 2020. [Online]. Available: http://discoverybrain.com/ediscovery /surprising-research-on-ocr-technologies/GitHub Repo of Tesseract. Accessed on: Oct 20, 2019. [Online]. Available: https://github.com/tesseract-ocr/tesseract

[15] “Accuracy and Performance of Tesseract 4.0”. Accessed on: Oct 8, 2019. [Online]. Available: https://github.com/tesseract-ocr/tesseract/wiki/4.0-Accuracy-and -Performance

[16] “The internal working of Tesseract”. Accessed on: Sep 30, 2019. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/ archive/.pdf

[17] “Comparison of OCR Engines”. Accessed on: Sep 15, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software.

[18] "Improving the quality of OCR output using image processing". Accessed on: Jan 21, 2020. [Online]. Available: https://github.com/tesseract-ocr/ tesseract/wiki/ ImproveQuality#page-segmentation-method.

[19] "Training Tesseract 4.0". Accessed on: Jan 12, 2020. [Online]. Available: https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00.

[20] "Tesseract Support". Accessed on: Jan 18, 2020. [Online]. Available: https:// github.com/rmtheis/tess-two/issues/196.

[21] "Tesseract 4.1.0". [Online]. Available: https://github.com/tesseract-ocr/ tesseract/archive/4.1.0.zip

[22] "Leptonica 1.78.0". [Online]. Available: http://www.leptonica.org/source/ leptonica-1.78.0.tar.gz

[23] "libjpeg v9c". [Online]. Available: https://github.com/Nodrev/libjpeg

[24] "libpng 1.6.37". [Online]. Available: https://sourceforge.net/projects/libpng /files/libpng16/1.6.37/

[25] "Tesseract v4.0.0 trained data files". [Online]. Available: https://github.com/ tesseract-ocr/tessdata_best

[26] "PDF Renderer". [Online]. Available: https://developer.android.com/reference /android/graphics/pdf/PdfRenderer

[27] "Tesseract4Android". [Online]. Available: https://github.com/adaptech-cz/ Tesseract4Android

Appendix A

Pseudocode

Client side:

```
/* Load image from gallery or take picture using camera  /*
image = imread("image file");

/* Select the language so that the required model can be loaded or downloaded if
required /*
lang = select_language();

/* Select text region of the image - Performed by cropping the image /*
image = select_text_region();

/*  If the preprocessing option is selected, send image to the server Uploading task
performed in background and not on the main thread  /*

if(processing_image() {
    /* Encoding the image so that it can be sent using JSON  /*
    image = encode(image);
    image = upload_to_server();
}

/* User chooses the layout between SIMPLE and COMPLEX */
layout = choose_layout();

/* Do layout analysis if user selects complex layout */
if(layout == COMPLEX) {
    image = analyze_layout(image);
    /* User selects text you want in the final document from the layout analyzed
    image */
    select_text_regions(image);
}
```

/* Select the language so that the required model can be loaded or downloaded if required */

lang = select_language();

/* OCR of the image /*

result = OCR(image, lang, ocr_engine_mode);

/* Convert the result in pdf or other required format /*

output = convert(image, reqd_format);

/* Save the doc/pdf to the external storage of the phone /*

save(output);

Server side:

json = get_json_from_client();

/* Decoding the image from received json /*

image = decode(json);

/* Rescaling the image to 300dpi /*

rescaledImage = rescale(image);

/* get grayscale image /*

grayscaleImage = convert(rescaledImage);

/* Removal of shadow from the image /*

imageWithoutShadow = shadowRemoval(deskewedimage);

/* Removal of border from the image /*

imageWithoutBorder = borderRemoval(imageWithoutShadow);

/* Skew correction of the image with the required angle /*

deskewedImage = deskew(grayscaleImage);

/* Binarising the image /*

binarisedImage = binarisation(imageWithoutBorder);

```
/* Checking the background color of the image /*
color = checkBackgroundColor(binarisedImage);

/* Inverting the image /*
invertedImage = Inversion(binarisedImage, color);

/* Downloading the image from the server to the client device /*
image = download(imageWithoutNoise);
```