Book Recognition System

Submitted in partial fulfillment of the requirements
of the degree of

T. E. Computer Engineering

Chelamallu Vinutha Roll No: 13 PID: 172123
Melita Coutinho Roll No: 15 PID:172069
Jenny D'cruz Roll No: 20 PID:172047

Guide(s):

Ms. Anuradha Sreenivasan
Asst. Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)

University of Mumbai
2019 - 2020

CERTIFICATE

This is to certify that the project entitled **"Book Recognition System"** is a bonafide work of **"Chelamallu Vinutha (Roll No: 13), Melita Coutinho (Roll No: 15) and Jenny D'cruz (Roll No: 20)"** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of T.E. in Computer Engineering.

(Ms. Anuradha Sreenivasan)
Guide

(Dr. Kavita Sonawane)
Head of Department

Project Report Approval for T.E.

This project report entitled ***Book Recognition System*** by **Ms. *Chelamallu Vinutha,* Ms. *Melita Coutinho* and Ms. *Jenny D'cruz*** is approved for the degree of ***T.E. in Computer Engineering.***

Examiners

1.----------

2.----------

Date:

Place: Mumbai

Declaration

We declare that this written submission represents my ideas in our words and where other's ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

Chelamallu Vinutha Roll No: 13

(Signature)

Melita Coutinho Roll No: 15

(Signature)

Jenny D'cruz Roll No: 20

Date:

Abstract

We need machine learning for tasks that are too complex for humans to code directly, i.e. tasks that are so complex that it is impractical, if not impossible, for us to work out all of the nuances and code for them explicitly. So instead, we provide a machine learning algorithm with a large amount of data and let it explore and search for a model that will work out what the programmers have set out to achieve. For example, it's very hard to write programs that solve problems like recognizing an object. In our application we recognize the images uploaded from a novel viewpoint, in new lighting conditions, in a cluttered scene. While we look for books online the details and genre of the book may be mentioned in the details and the authors name is skipped out on. We don't know what program to write because we don't know how it's done in our brain. Even if we had a good idea for how to do it, the program might be horrendously complicated. For this we use image recognition. Image recognition, in the context of ML, is the ability of software to identify objects, places, people, writing and actions in images. It is used to perform a large number of machine-based visual tasks, such as labeling the content of images with meta-tags.

Contents

Chapter		Title	Page No.
1		**INTRODUCTION**	**1**
	1.1	Project Description	1
	1.2	Problem Formulation	1
	1.3	Motivation	1
	1.4	Proposed Solution	1
	1.5	Scope of the project	2
2		**REVIEW OF LITERATURE**	**3**
	2.1	Overview of Tesseract OCR Engine	3
	2.2	Book Cover Recognition	3
	2.3	Smart Library	3
3		**SYSTEM ANALYSIS**	**5**
	3.1	Functional Requirements	5
	3.2	Non Functional Requirements	5
	3.3	Specific Requirements	6
	3.4	Use-Case Diagrams and description	7
4		**ANALYSIS MODELING**	**10**
	4.1	Activity Diagrams / Class Diagram	10
	4.2	Functional Modeling	12
5		**DESIGN**	**14**
	5.1	Workflow	14
	5.2	Technology used	15
	5.2.1	Data Preprocessing: Adaptive Thresholding	15
	5.2.2	Component Analysis	16
	5.2.3	Words detection, paragraph lines	16
	5.2.4	Two Steps Recognition	16

	5.3	User Interface	18
6		**IMPLEMENTATION**	**21**
	6.1	Algorithms / Methods Used	21
	6.1.1	Preprocession: Adaptive Thresholding	21
	6.2	Working of the project	22
7		**CONCLUSIONS**	**25**
8		**REFERENCES**	**26**
9		**ACKNOWLEDGEMENTS**	**27**

List of Figures

Fig. No.	Figure Caption	Page No.
1.	Use Case Diagram of Book cover Recognition System	7
2.	Class Diagramof Book cover Recognition System	10
3.	Activity Diagram for Book cover Recognition System	11
4.	Level 0 DFD	12
5.	Level 1 DFD	12
6.	Level 2 DFD	13
7.	OCR Process workflow	14
8.	Tesseract OCR process	15
9.	Main Window	18
10.	Uploading the input image	18
11.	Input image	19
12.	Image to text	19
13.	About us page	20

List of Abbreviations

Sr. No.	Abbreviation	Expanded form
1.	DFD	Data Flow Diagram
2.	OCR	Optical Character Recognition
3.	LSTM	Long Short term memory

Chapter 1

Introduction

1.1 Project Description

It is observed that when we search for books online the details and genre of the book may be mentioned in the details and the author's name is skipped out on. The issue faced is font not being recognised on certain books, the authors name is not recognised off of the cover. Even if we had a good idea for how to do it, the program might be complicated. For this we use image recognition techniques to get the name of the book and author by uploading a book cover.

1.2 Problem Formulation

Software systems face problems reading characters, numbers from an image which makes it difficult to get a proper output. Hence we prepared a web application which reads the capital letters, small letters, and special characters on a book cover. Thus displaying the text off of the book from the cover image uploaded and this makes random book recognition online easier.

1.3 Motivation

Image recognition is one the most useful means to search anything on the search engines. Especially when it can read the text in it. When we search for any books online we usually skip the name of the author due the font. Thus we make a system which detects the name of the book and the author just by uploading the image of a book cover. The book recognition system is a necessity for the smooth functioning of book sales and as it has potential application areas which would reduce the task of data entry.

1.4 Proposed Solution

The book recognition system detects the characters, numbers, special characters, etc. from a book cover which is uploaded as displays the name of the book and the author. The user

uploads a picture of the book cover. The system checks for the existence of this book cover in its dataset and if it finds the book, it displays the characters from its cover to the user. To detect the same we use Python-tesseract. Python-tesseract is an optical character recognition (OCR) tool for python. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. That is, it will recognize and "read" the text embedded on the book cover. Then we used flask which is a web framework in Python. Flask provides tools, libraries and technologies that we used to build our web application for book recognition systems.

1.5 Scope of The Project

The scope of our model, Book Cover Recognition is to provide an efficient and enhanced application for the users to perform character recognition. Other than that the book recognition system can be used to get an introduction or summary of a book whose book cover is scanned. This will be helpful for libraries. As in when a reader wishes to read a book one can simply scan the book cover and get the name of the book, name of the author, it's genre and introduction or summary of the book. This will help the reader to choose the book we want to read without actually open the book by reading its details on the screen.

Chapter 2

Review of Literature

2.1 Overview of Tesseract OCR Engine:

Authors: Akhil Nair

Publications: National Institute of Technology Calicut

Approach:

The author gives the overall working of the tesseract OCR. He explains the architecture of tesseract OCR, whose steps include adaptive thresholding,page layout analysis,detection of baseline and words and recognizing the word.

2.2 Book Cover Recognition

Authors: Linfeng Yang(linfeng@stanford.edu) , Xinyu Shen(xinyus@stanford.edu)

Publications: web.stanford.edu

Approach:

The author proposes to find the cost, reviews and more information of the book just by uploading the image which will help a new reader to decide whether to buy a particular book or not. The author uses OCR techniques to detect the name of the book and the author. Then uses these as keywords to find other details in search engines.

2.3 Smart Library: Identifying Books on Library Shelves using Supervised Deep Learning for Scene Text Reading

Authors: Xiao Yang, Dafang He, Wenyi Huang, Alexander Ororbia Zihan Zhou, Daniel Kifer, C. Lee Giles

Publications: *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017*, 245-248, 2017.

Approach:.

Chapter 2: Review of Literature

In large libraries it is difficult to find a particular book from the book shelves. Hence, the author suggests an alternative to find the book by a deep neural network-based system to automatically localize, recognize and index text on bookshelves images. The author first processes bookshelves images to localize and recognize book spine text so as to build a digital book inventory. Then, they utilized this digital inventory to help users quickly locate a book or volume they are looking for. They take an original image and rotate it based on estimation of dominant orientation. Then the author salience image for better detection of book spine. The image is then segmented into different book spines and then detect lines of words. These words are signal processed to find the query keywords to find the corresponding ISBN number. Using this number the location of the stack of the book belongs to is determined and thus the location of the book on the shelf.

Chapter 3

System Analysis

3.1 Functional Requirements

Major functionalities associated with the user are:

- Enable users to upload the image of the book cover.

Major functionalities associated with the system are as follows:

- System preprocesses the image by applying the concept of adaptive binarization.
- System does component analysis on preprocessed images
- System detects the text on the image of the cover uploaded.
- System makes substrings of the text it recognizes.
- These substrings should be used by the system to find the closest match in the dataset which consists of the name of the book and author and display the same as the output.

Interface Requirements:

1. User needs to upload an image (System only accepts ‘png’,’jpg’,jpeg’)
2. Details about the system and the students are to be displayed to the user in the “About us” page.

Business Requirements:

1. Image must be uploaded before a request is submitted.
2. Clicking the submit button we get the output.

3.2 Non Functional Requirements

3.2.1 Performance

The computer running the software must have a powerful CPU and GPU so that the data can be processed faster and there won’t be lag in the entire process of detect the text in the image

3.2.2 Reliability

The system takes in the inputs without any error and determines the text on the book cover which is uploaded and then refines the text ,thus displaying the name of the book and it's author.

3.2.3 Usability

The system is easy to handle and navigates in the most expected way with no delays. The system interacts with the user in a very friendly manner making it easier for the users to use the system.

3.3 Specific Requirements

3.3.1 User Interfaces

- Front-end: HTML, CSS, JavaScript, Web browser, Flask
- Back-end: Python

3.3.2 Hardware Requirements

- CPU Type : Intel i3 or above
- Clock speed : 2.1GHz
- Ram size : 4GB and above
- Hard Disk capacity : 100GB and above
- Working keyboard

3.3.2 Software Requirements

- Operating System : Windows
- Python 3.5 or above

3.3.3 Communication Interfaces

This system will be completely based on a local system of the user.

3.4 Use-Case Diagrams and Description

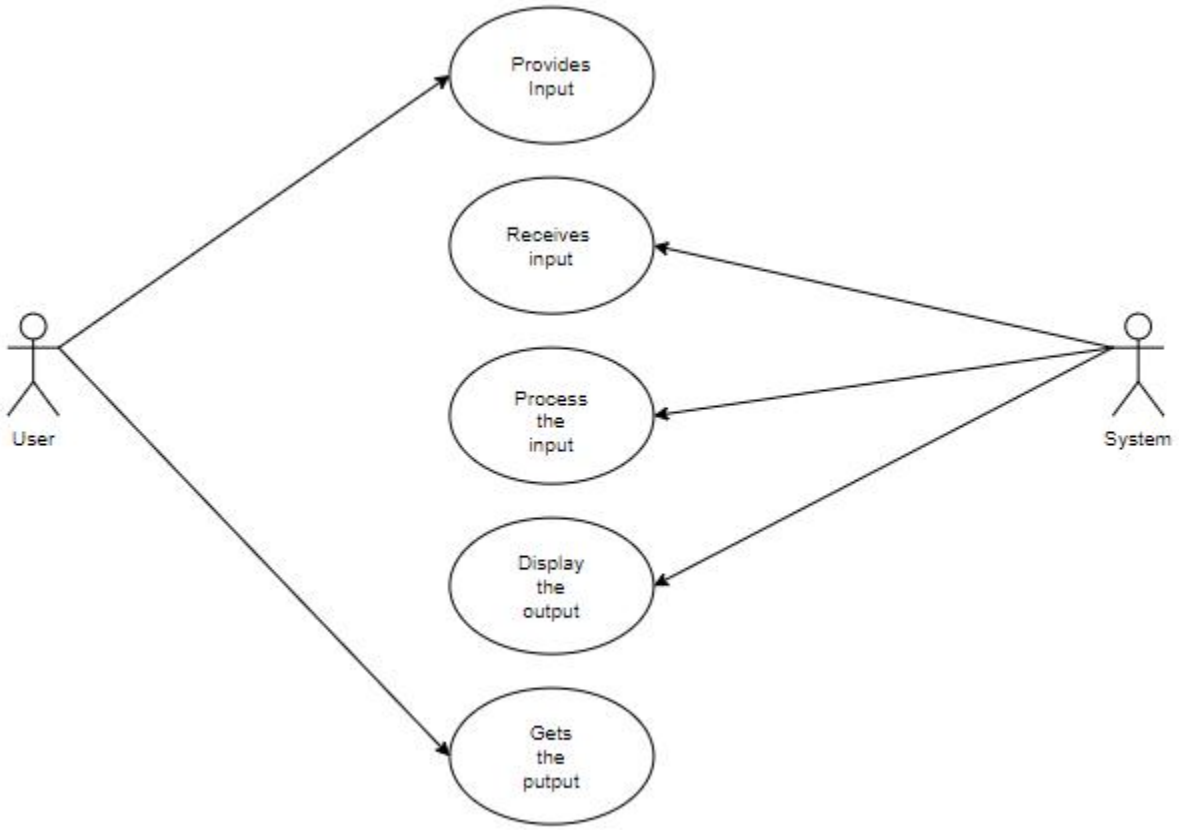

Fig 1. Use Case Diagram of Book cover Recognition System

Use Case Diagram:

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and the use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a “System” is something being developed and operated, such as a website. The “Actors” are people or entities operating under defined roles within the system.

Use Case Specifications:

- **Use case:** Provides Input

  Brief Description: This use case will expect the user to provide an image with extensions of jpg,jpeg and png only as an input to the system.

  Primary Actor: User

  Main Flow:

  1. On the first page users will be a choose file option where they have to upload the image of the book cover they want to detect the text in.
  2. Then click on the upload option to get the text on the book cover

- **Use Case:** Receives and stores input

  Brief Description: This use case will receive the input given by the user.

  Primary Actor: System

  Main Flow:

  1. The image of the book cover given will be retrieved as an input by the system.
  2. After retrieving the input, the system will send it for further processing.

- **Use Case:** Processes the input

  Brief Description: In this use case the system will preprocess the image and then differentiate between the area where there is text and where there is not.

  Primary Actor: System

  Main Flow:

  1. It will first convert the image in binary representation.
  2. Then detect the text and the non text area

3. Organizes text lines into blobs and then the text lines are broken into words depending on the spacing between them.
4. Then detects each word.

- **Use Case:** Display output

  Brief Description: The system will display the output to the user.

  Primary Actor: System

  Main Flow:

  1. The system will display the text present on the book cover.

  Chapter 3 System Analysis

- **Use Case:** Gets output

  Brief Description: The user can view the output on the screen

  Primary Actor: User

  Main Flow:

  1. After the processing is done by the system, it will generate an output.
  2. This output generated can be viewed by the users on the window.

Chapter 4

Analysis Modeling

4.1 Class Diagram and Activity Diagram

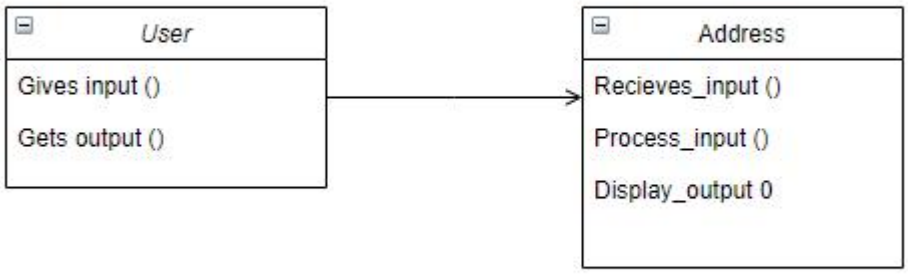

Fig 2. Class Diagram of Book cover Recognition System

Chapter 4: Analysis Modeling

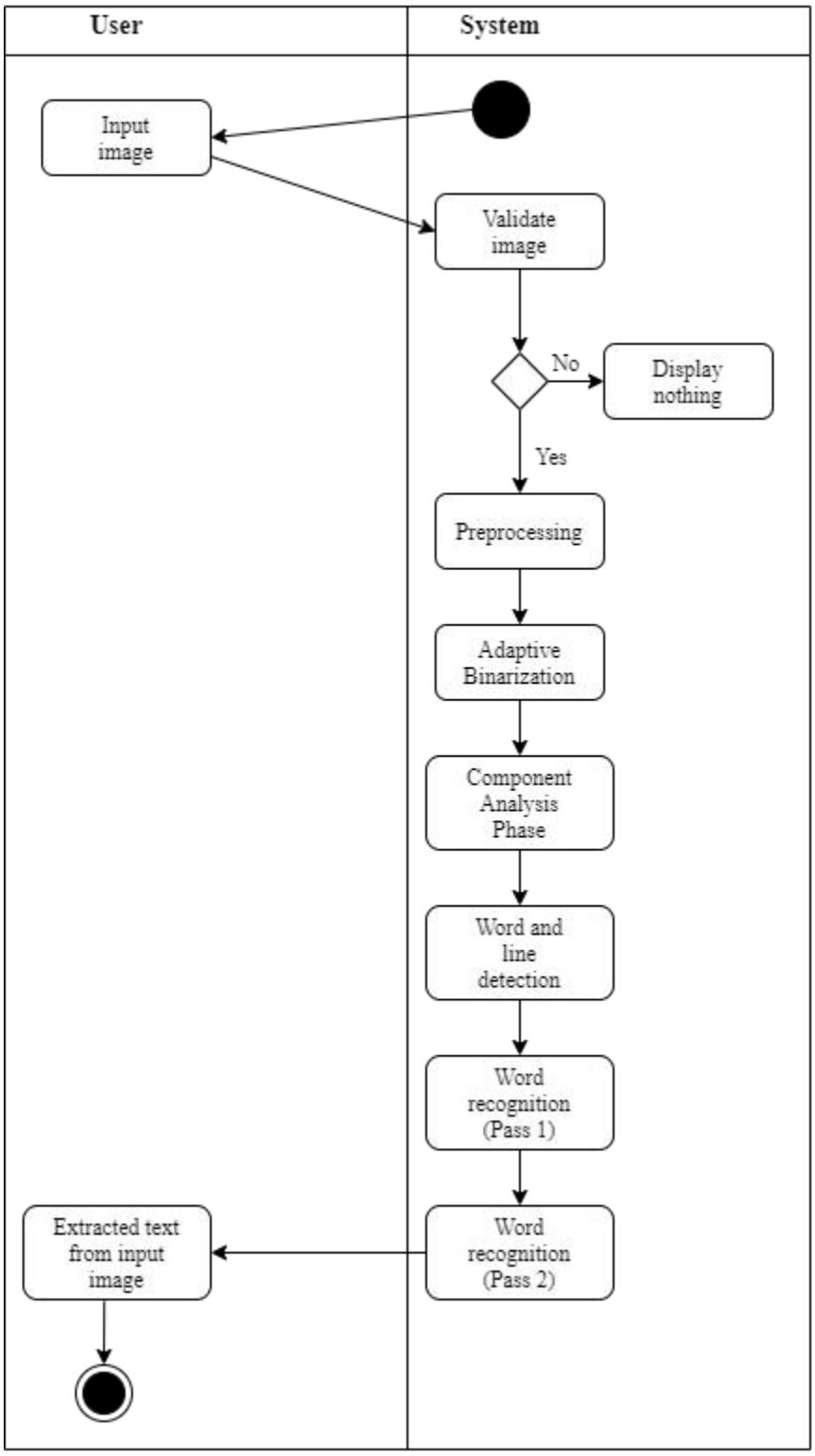

Fig 3. Activity Diagram for Book cover Recognition System

4.2 Functional Modelling

Data Flow Diagram

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on flow of information, where data comes from, where it goes and how it gets stored.

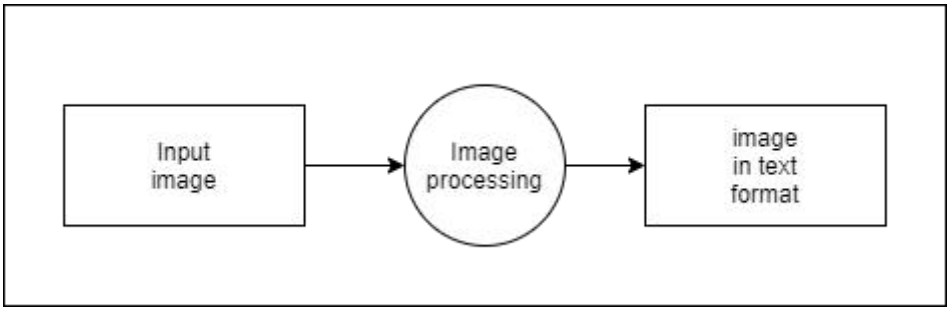

Fig 4. Level 0 DFD

Level 0 DFD:

DFD Level 0 provides a more detailed breakout of pieces of Context Level DFD. It basically explains the system more in detail.

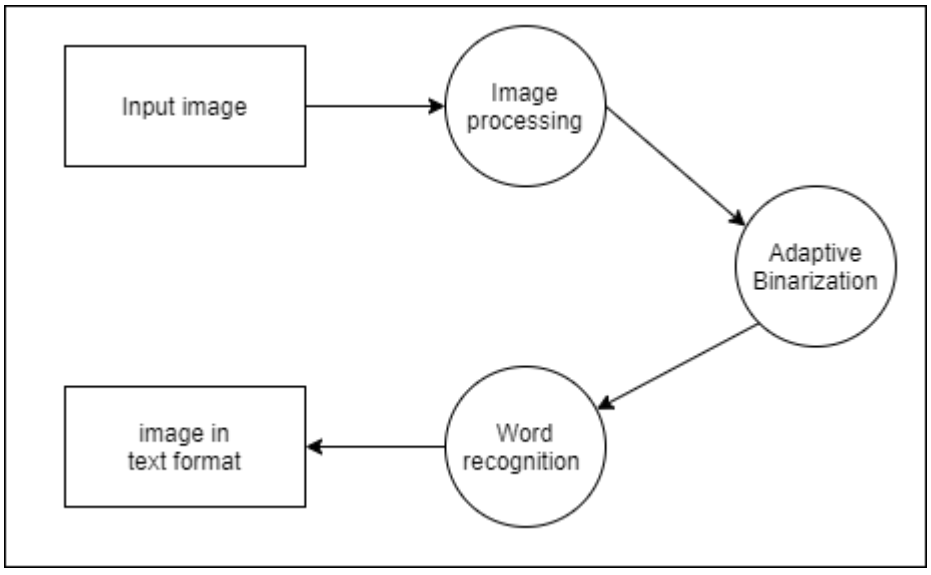

Fig 5. Level 1 DFD.

Level 1 DFD:

In this level, the level the Level 0 is further explained in detail.

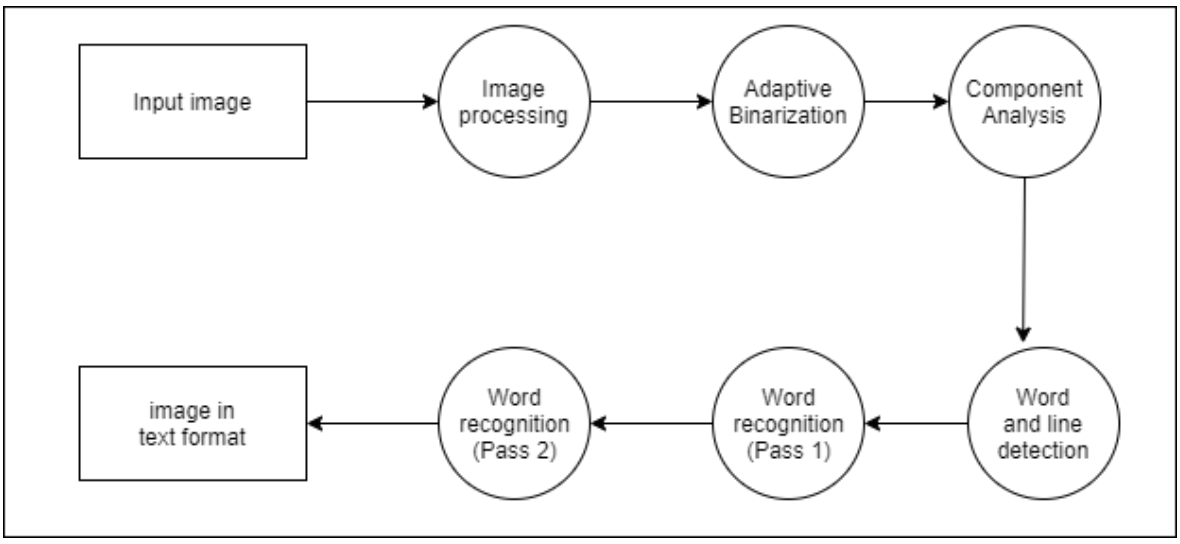

Fig 6. Level 2 DFD.

Level 2 DFD:

In this level, the level the Level 0 is further explained in detail.

Chapter 5

Design

5.1 Workflow:

- Tesseract is an open source text recognition (OCR) Engine, available under the Apache 2.0 license. It can be used directly, or (for programmers) using an API to extract printed text from images. It supports a wide variety of languages.
- Tesseract doesn't have a built-in GUI, but there are several available from the 3rd Party page. Tesseract is compatible with many programming languages and frameworks through wrappers that can be found here.
- It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

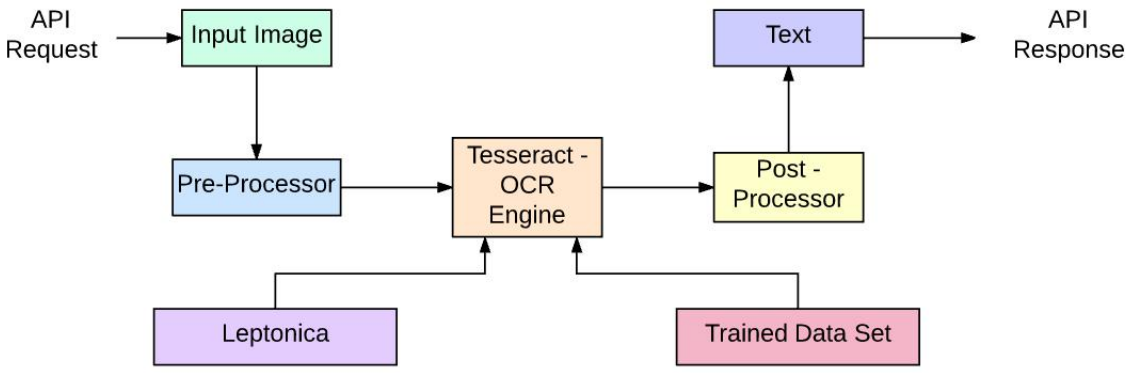

Fig 7. OCR Process workflow

- Tesseract includes a new neural network subsystem configured as a text line recognizer.
- It has its origins in OCRopus' Python-based LSTM implementation but has been redesigned for Tesseract in C++.
- The neural network system in Tesseract pre-dates TensorFlow but is compatible with it, as there is a network description language called Variable Graph Specification Language (VGSL), that is also available for TensorFlow.

- To recognize an image containing a single character, we typically use a Convolutional Neural Network (CNN). Text of arbitrary length is a sequence of characters, and such problems are solved using RNNs and LSTM is a popular form of RNN.

5.2 Technology used:

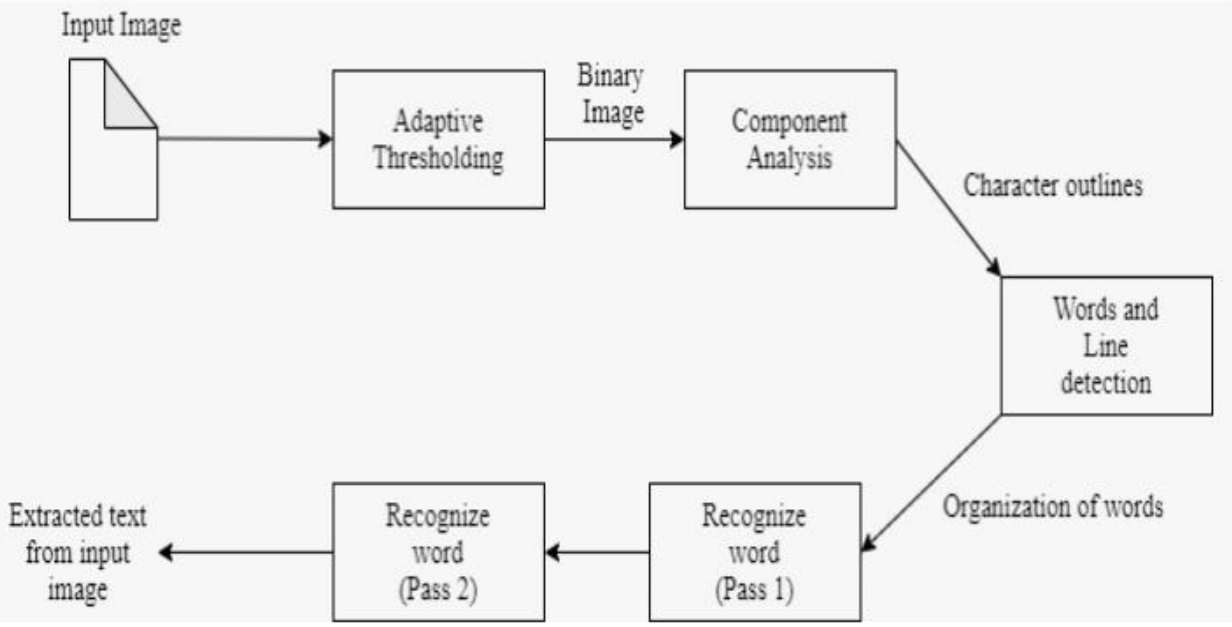

Fig 8. Tesseract OCR process

5.2.1 Data Preprocessing: Adaptive Thresholding

If the value of the current pixel is the percent lower than the average then it is set to black, otherwise it is set to white. This method works because comparing a pixel to the average of nearby pixels will preserve hard contrast lines and ignore soft gradient changes. The advantage of this method is that only a single pass through the image is required. Wellner uses 1/8th of the image width for the value of s and 15 for the value of t. However, a problem with this method is that it is dependent on the scanning order of the pixels. In addition, the moving average is not a good representation of the surrounding pixels at each step because the neighbourhood samples are not

evenly distributed in all directions. By using the integral image (and sacrificing one additional iteration through the image), we present a solution that does not suffer from these problems.

This technique is clean, straightforward, easy to code, and produces the same output independently of how the image is processed. Instead of computing a running average of the last s pixels seen, we compute the average of an s x s window of pixels centered around each pixel. This is a better average for comparison since it considers neighbouring pixels on all sides. The average computation is accomplished in linear time by using the integral image. We calculate the integral image in the first pass through the input image. In a second pass, we compute the s x s average using the integral image for each pixel in constant time and then perform the comparison. If the value of the current pixel is the percent less than this average then it is set to black, otherwise it is set to white.

5.2.2 Component Analysis

Tesseract therefore assumes that its input is a binary image with optional polygonal text regions defined. Processing follows a traditional step-by-step pipeline, but some of the stages were unusual in their day, and possibly remain so even now. The first step is a connected component analysis in which outlines of the components are stored. This was a computationally expensive design decision at the time, but had a significant advantage: by inspection of the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text. Tesseract was probably the first OCR engine able to handle white-on-black text so trivially. At this stage, outlines are gathered together, purely by nesting, into Blobs.

5.2.3 Words detection, paragraphs lines

Blobs are organized into text lines, and the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is

broken into words using definite spaces and fuzzy spaces. Recognition then proceeds as a two-pass process.

5.2.4 Two Steps Recognition

- **First pass:** An attempt is made to recognize each word in turn by the Static Character Classifier. Since the classifier is able to recognize damaged characters easily, the classifier was not trained on damaged characters. In fact, the classifier was trained on a mere 20 samples of 94 characters from 8 fonts in a single size, but with 4 attributes (normal, bold, italic, bold italic), making a total of 60160 training samples. This is a significant contrast to other published classifiers, such as the Calera classifier with more than a million samples , and Baird's 100-font classifier with 1175000 training samples.

- **Second pass:** The text goes through the adaptive classifier. It has been suggested and demonstrated that OCR engines can benefit from the use of an adaptive classifier. Since the static classifier has to be good at generalizing to any kind of font, its ability to discriminate between different characters or between characters and non-characters is weakened. A more font-sensitive adaptive classifier that is trained by the output of the static classifier is therefore commonly used to obtain greater discrimination within each document, where the number of fonts is limited. Tesseract does not employ a template classifier, but uses the same features and classifier as the static classifier.

The only significant difference between the static classifier and the adaptive classifier, apart from the training data, is that the adaptive classifier uses isotropic baseline/x-height normalization, whereas the static classifier normalizes characters by the centroid (first moments) for position and second moments for anisotropic size normalization. The baseline/x-height normalization makes it easier to distinguish upper and lower case characters as well as improving immunity to noise specks. The main benefit of character moment normalization is removal of font aspect ratio and some degree of font stroke width. It also makes recognition of sub and

superscripts simpler, but requires an additional classifier feature to distinguish some upper and lower case characters.

5.3 User Interface

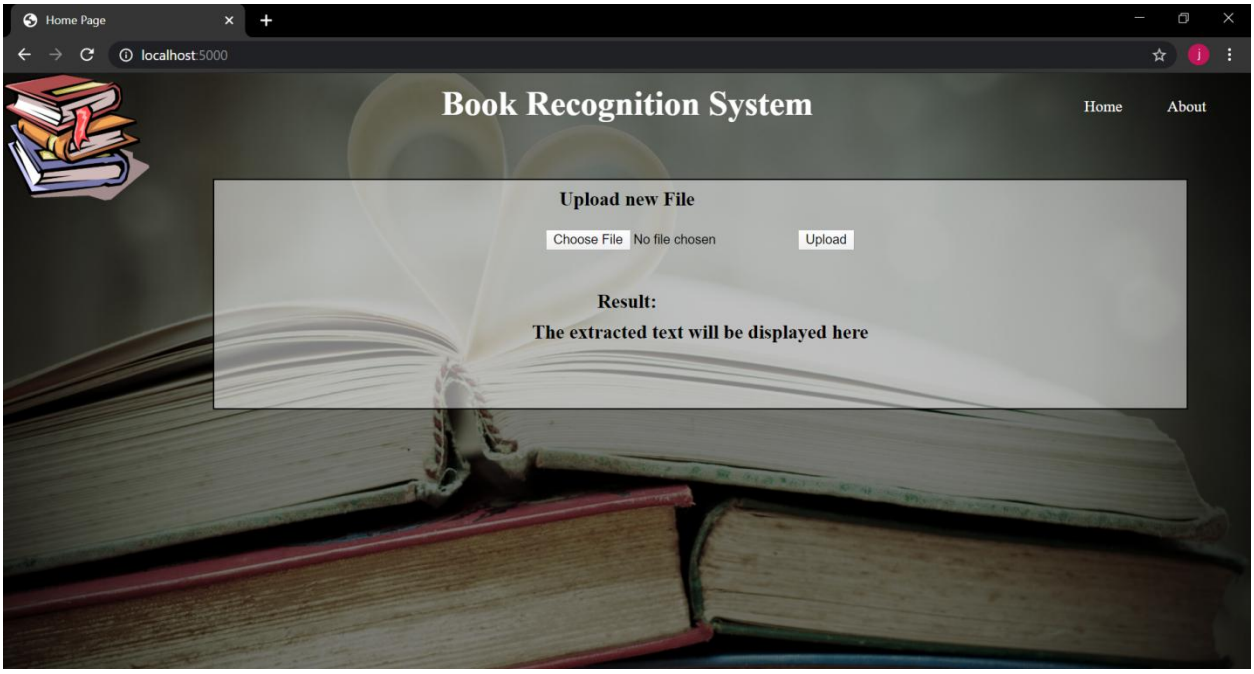

Fig 9. Main Window

Chapter 5: Design

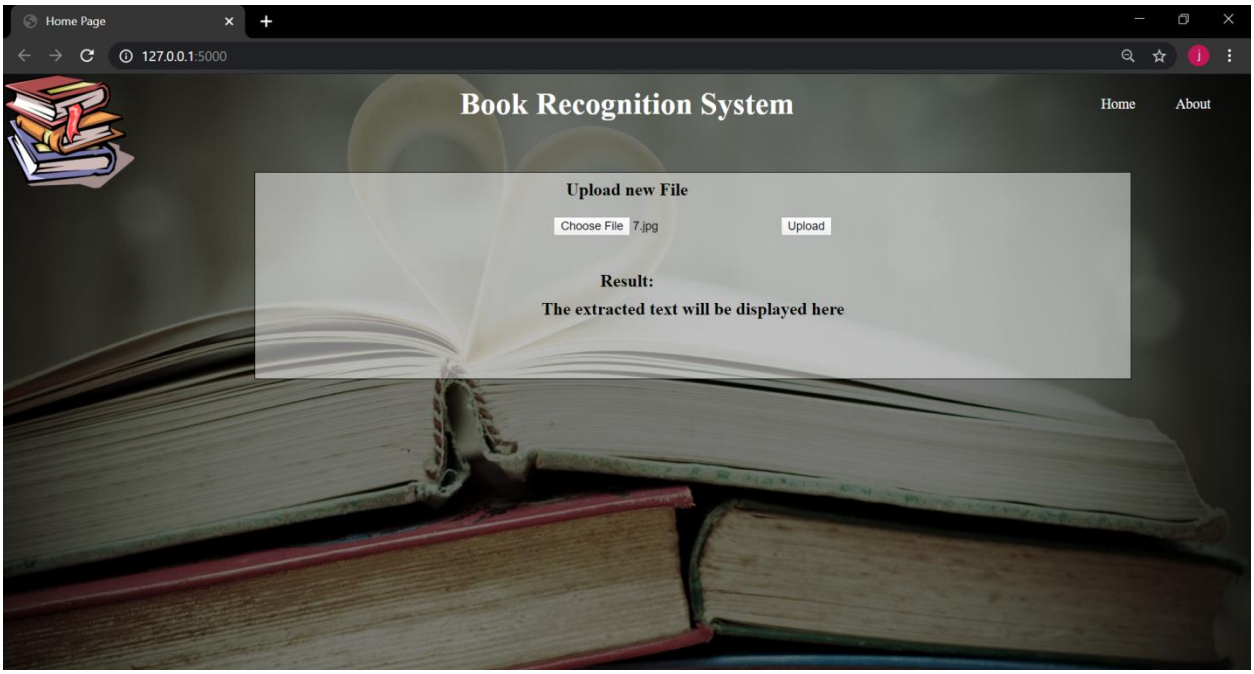

Fig 10. Uploading the input image

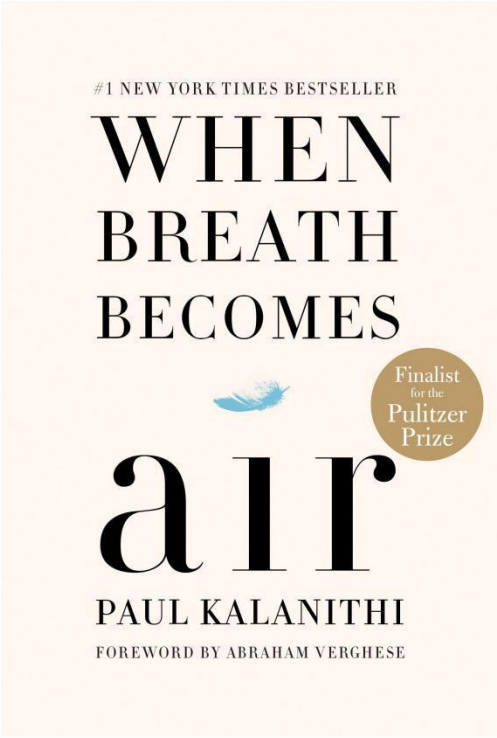

Fig 11. Input image

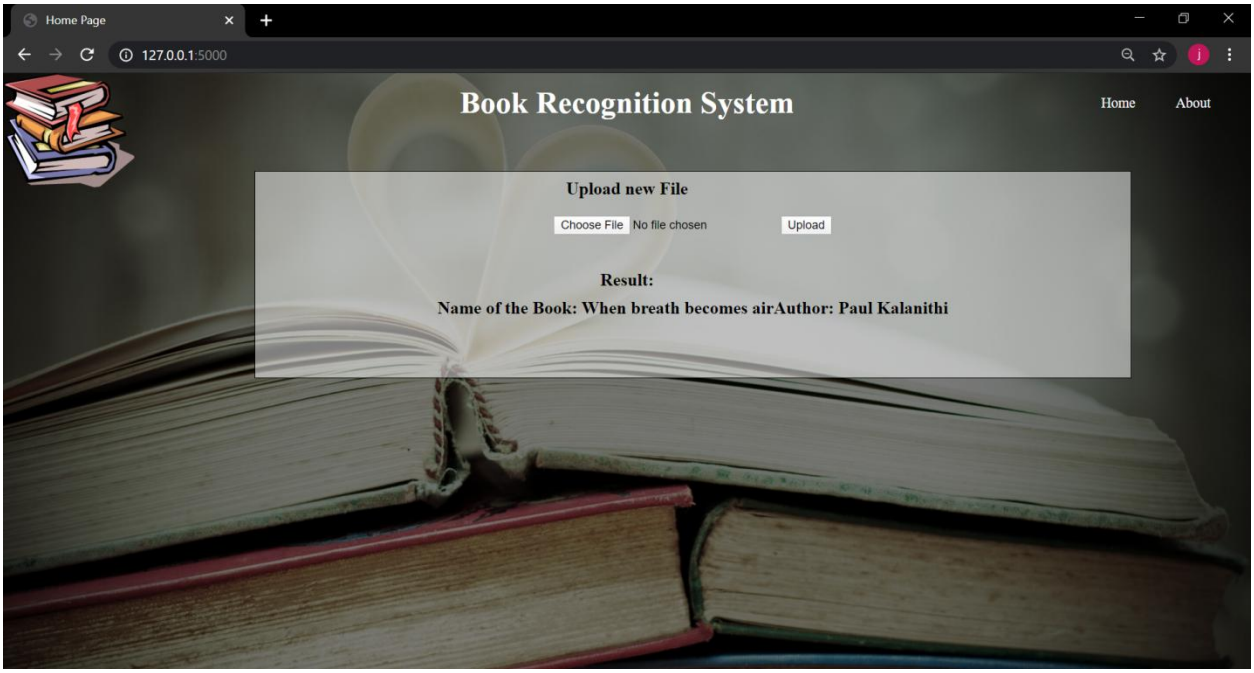

Fig 12. Image to text

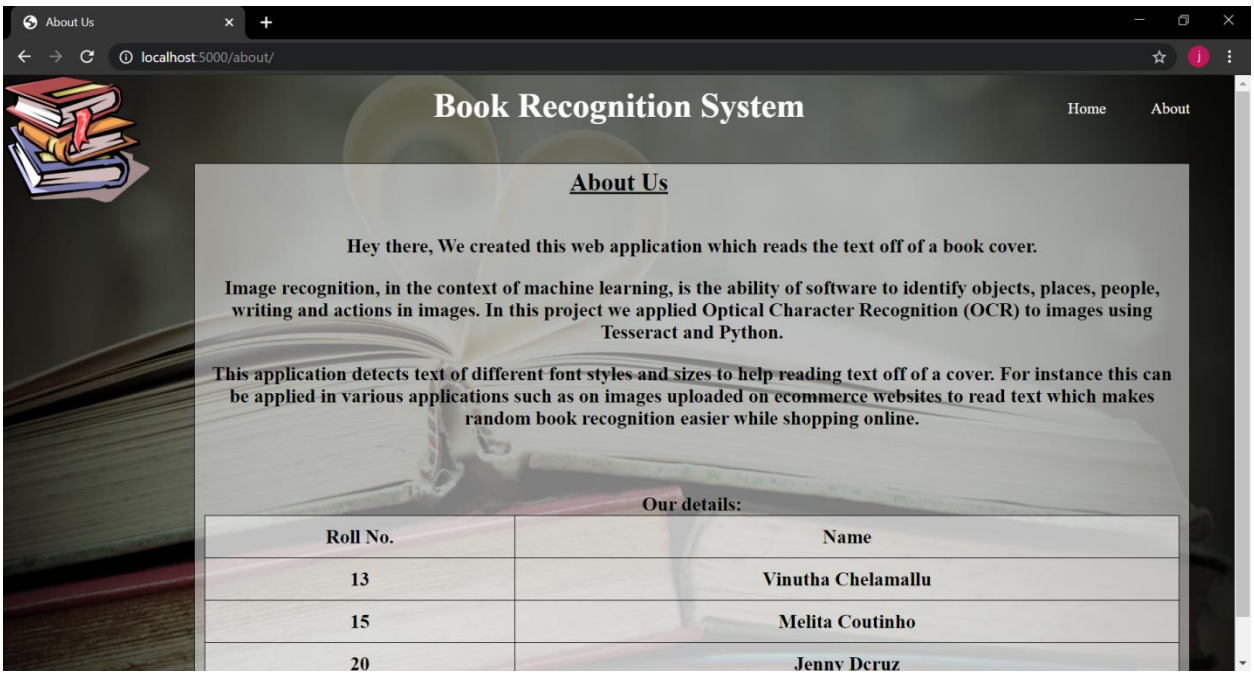

Fig 13. About us page

Chapter 6

Implementation

6.1 Algorithm Used

6.1.1 Preprocessing: Adaptive Thresholding

The following pseudocode demonstrates adaptive thresholding technique for input image in, output binary image out, image width w and image height h:

```
procedure AdaptiveThreshold(in,out,w,h):
for i = 0 to w do
    sum ← 0
  for j = 0 to h do
        sum ← sum + in[i,j]
        if i = 0 then
            intImg[i,j] ← sum
        else
            intImg[i,j] ← intImg[i - 1,j] + sum
        end if
    end for
end for
for i = 0 to w do
    for j = 0 to h do
        x1← i - s/2 {border checking is not shown}
        x2 ← i + s/2
        y1 ← j - s/2
        y2 ← j + s/2
        count ← (x2 - x1) x (y2 - y1)
        sum ← intImg [x2,y2] - intImg[x2, y1 - 1] - intImg[x1 - 1, y2] + intImg[x1 - 1, y1 - 1]
        if(in[i,j] x count) < = (sum x (100 - t)/100) then
```

```
            out[i, j] ← 0
        else
            out[i, j] ← 255
        end if
    end for
end for
```

6.2 Working of the project

CODE SNIPPETS

Working of OCR using tesseract and python:

```
def ocr_core(filename):
    extractedInformation = tess.pytesseract.image_to_string(Image.open(filename))
    text = extractedInformation.lower().replace(',','').replace(' ','').replace('_','').replace(':','')
                        .replace('=','').replace('(','').replace(')','').replace('"','').replace('|','')
                        .replace('#','').replace('$','').replace('&','').replace('0','').replace('1','')
                        .replace('2','').replace('3','').replace('4','').replace('5','').replace('6','')
                        .replace('7','').replace('8','').replace('9','')
    text = text.translate({ord(c): None for c in string.whitespace})
    with open('tess.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            for col in row:
                if col==text:
                    return("\nName of the Book: "+row[1]+"Author: "+row[2])
```

Main file linking the ocr function, upload page and about us page:

```
@app.route('/')
def home():
    return render_template('upload.html')

# define a folder to store and later serve the images
UPLOAD_FOLDER = '/static/uploads/'

# allow files of a specific type
ALLOWED_EXTENSIONS = set(['jfif','png', 'jpg', 'jpeg'])

# function to check the file extension
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

```python
@app.route('/',methods=['GET', 'POST'])
def upload_page():
    if request.method == 'POST':
        # check if there is a file in the request
        if 'file' not in request.files:
            return render_template('upload.html', msg='No file selected')
        file = request.files['file']
        # if no file is selected
        if file.filename == '':
            return render_template('upload.html', msg='No file selected')

        if file and allowed_file(file.filename):

            # call the OCR function on it
            extracted_text = ocr_core(file)

            # extract the text and display it
            return render_template('upload.html',extracted_text=extracted_text,img_src=UPLOAD_FOLDER + file.filename)
    elif request.method == 'GET':
        return render_template('upload.html')

@app.route('/about/')
def about():
    return render_template('about.html')

if __name__ == '__main__':
    app.run()
```

Upload page body

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Home Page</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>

  <body>
    {% extends "index.html" %}
    {% block content %}

      {% if msg %}
      <h1>{{ msg }}</h1>
      {% endif %}

      <div class="table">
      <table role="presentation" border="0"  cellpadding="0" align="center" width="100%">
        <tr>
        </br></br></br></br></br></br>
          <th><br><font size="5" color="black">

          <div class="upload">
            <h1>Upload new File</h1>
          </div>
          <form method=post enctype=multipart/form-data>
            <p><input type=file name=file>
            <input type=submit value=Upload>
          </form>
          <br>
          <div class="upload">
            <h1>Result:</h1>
          </div>
```

```
            <div class="text">
            {% if extracted_text %}
              <p><b> {{ extracted_text }} </b></p>

            {% else %}
              <p>The extracted text will be displayed here </p>
            {% endif %}
          </br></br>
          </div></font>
          </th>
        </tr>
      </table>
      </div>
      {% endblock %}
    </body>
  </html>
```

Chapter 7

Conclusion

Analysis of Results:

We used Logistic Regression, Random Forest and Support Machine Vector to find the accuracy of our model. Accuracy given by each of the above given algorithms.

Name of the Algorithm	Accuracy Obtained (%)
Random Forest	87.5
Logistic Regression	81.8
Support Vector Machine	54.54

As given above Random Forest Algorithm gave the highest accuracy among all.

Confusion Matrix:

Predicated Efficiency Actual Efficiency	Text Detected Incorrectly	Text Detected Correctly
Text Detected Incorrectly	5	1
Text Detected Correctly	0	2

Conclusion:

Tesseract is now behind the leading commercial engines in terms of its accuracy. Its key strength is probably its unusual choice of features. The main requirement of the book recognition system was tesseract OCR which detected the text in the image. By using the book recognition we make it easier for the reader to know the name of the author and the book easily. This project will also help to find details of a book over the Internet. Further this project can further be used to know the synopsis of the book just by scanning the book cover.

References

[1] Xiao Yang, Dafang He, Wenyi Huang, Alexander Ororbia, Zihan Zhou, Daniel Kifer, C. Lee Giles, "Smart Library: Identifying Books on Library Shelves Using Supervised Deep Learning for Scene Text Reading," *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017*, 245-248, 2017.

[2] Linfeng Yang(linfeng@stanford.edu) , Xinyu Shen(xinyus@stanford.edu), "Book cover recognition".

[3]] Flask,Available at: https://www.fullstackpython.com/flask.html

[6] DBSCAN, s. (2019). *scikit-learn: clustering text documents using DBSCAN*. [online] Stack Overflow. Available at:https://stackoverflow.com/questions/25217065/scikit-learn-clustering-text-documents-using-dbscan [Accessed 15 Apr. 2019].

[7] (@brandonmrose), B. (2019). Document Clustering with Python. [online] Brandonrose.org. Available at: http://brandonrose.org/clustering_mobile [Accessed 15 Apr. 2019].

[8]Badla, Sahil, "IMPROVING THE EFFICIENCY OF TESSERACT OCR ENGINE" (2014). *Master's Projects*. 420.

[9]Ray Smith , "An Overview of the Tesseract OCR Engine"in proceedings of document analysis and recognition.. icdar 2007. In IEEE Ninth International Conference, 2007.

[10]Akhil s Nair,"Overview of Tesseract OCR engine",National Institute of Technology Calicut

Acknowledgements

We take the opportunity to thank all those people who have helped and guided us through this project and make this experience worthwhile for us. We wish to sincerely thank our reverend **Bro. Jose Thuruthiyil** and principal **Dr. Sincy George** for giving us this opportunity for making a project in the Final Year of Engineering. We would also like to thank HOD of the Computer department **Dr. Kavita Sonawane** and all teaching and non teaching staff for their immense support and cooperation. Last but not the least we would like to thank **Mrs. Anuradha Sreenivasan** for guiding us throughout the project and encouraging us to explore this domain.