HAND WRITTEN TEXT RECOGNITION

A PROJECT REPORT

OF PROJECT-2 (IT 892)

BACHELOR OF TECHNOLOGY
In
Information Technology

(From **Maulana Abul Kalam Azad University of Technology, West Bengal**)

SUBMITTED BY

Shashikant Pathak (13000215096)

Shrikant Kumar (13000215103)

Sumit Kumar Trivedi (13000215114)

Vivek Kumar Kushwaha (13000215123)

Under the Guidance of

Prof. Piyal Sarkar (HOD)

Department of Information Technology
Techno Main Salt Lake
Kolkata-700091

Department of Information Technology
Techno Main, Salt Lake
EM-4/1, Salt Lake, Sec V, Kolkata-700091

BONAFIDE CERTIFICATE

Certified that this report for the project titled "**HAND WRITTEN TEXT RECOGNITION**" is a part of the final year project work being carried out by "**SHASHIKANT PATHAK , SHRIKANT KUMAR , SUMIT KUMAR TRIVEDI , VIVEK KUMAR KUSHWAHA**" as partial fulfillment for the degree of Bachelor of Technology in Information Technology, Maulana Abul Kalam University of Technology, West Bengal, under my supervision.

Full Signature of the Candidates (with date)

1 ----------

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

3. ----------

4. ----------

(Signature Of Supervisor)

(Signature of the Head of the Department)

ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our respected and esteemed mentor **Prof. Piyal Sarkar** of the Dept. of Information Technology, Techno Main, Salt Lake, for his valuable guidance, encouragement and helping us to pursue with our project work. His useful suggestions for this work and co-operative behavior are sincerely acknowledged.

We would like to express our sincere thanks to the teaching fraternity of the Department of Information Technology, for giving us this opportunity to undertake this project and also supporting us whole heartedly.

We also wish to express our gratitude to the all our teachers of the Department of Information Technology for their kind hearted support, guidance and utmost endeavor to groom and develop our academic skills.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during the effort in shaping this concept till now.

Full Signature of the Candidates (with date)

1. _____

2. _____

3. _____

4. _____

ABSTRACT

This project is based on off-line hand written text recognition system using *PyTesseract* library which is a wrapper for Google's Tesseract-OCR Engine, *Flask web framework, Pipenv and* Pillow library which is a fork of the *Python Imaging Library* (PIL).

We will use the *Flask web framework* to create our simple OCR server where we can upload photos for character recognition purposes. *Pipenv* since it also handles the virtual-environment setup and requirements management. Pillow library which is a fork of the *Python Imaging Library* (PIL) to handle the opening and manipulation of images in many formats in Python.

To implement this project we first, create a Flask application to act as an interface which will be used to upload image and in next step we create the script which will recognize text in our uploaded image.

TABLE OF CONTENTS

Chapter – 1

INTRODUCTION

1.1 Before We Start

In the running world, there is growing demand for the software systems to recognize characters in computer system when information is scanned through paper document. as we know that we have number of newspapers and books which are in printed format related to different subjects. These days there is a huge demand in “storing the information available in these paper documents in to a computer storage disk and then later reusing this information by searching process” . One simple way to store information in these paper documents in to computer system is to first scan the documents and then store them as images. But to reuse this information it is very difficult to read the individual contents and searching the contents form these documents line-by-line and word-by-word. The reason for this difficulty is the font characteristics of the characters in paper documents are different to font of the characters in computer system. As a result computer is unable to recognize the characters while reading them. This concept of storing the contents of paper documents in computer storage place and then reading and searching the content is called Document Processing. Sometimes in this document processing we need to process the information that is related to languages other than the English in the world. For this document processing we need a software system called *Character Recognition System*. This process is also called Document Image Analysis (DIA).
Thus our need is to develop character recognition software system to perform Document Image Analysis which transforms documents in paper format to electronic format. For this process there are various techniques in the world. Among all those techniques we have chosen Optical Character Recognition as main fundamental technique to recognize characters

As humans can understand the contents of an image simply by looking. Computers don't work the same way. They need something more concrete, organized in a way they can understand.

This is where Optical Character Recognition (OCR) needed. Whether it's recognition of car plates from a camera, or handwritten documents that should be converted into a digital copy, this technique is very useful. While it's not always perfect, it's very convenient and makes it a lot easier and faster for some people to do their jobs. We will build a simple script in Python that will help us detect characters from images and expose this through a Flask application for a more convenient interaction medium.

1.2 Motivation

Last year when i was preparing for my semester exam, it was needed some class notes to prepare for it. I asked my friend to send notes to prepare for examination. He sent it but his writing was not clear and it was difficult to understand. That time i wish if there were software so that i could convert it into standard writing style i.e computerized writing style so that anyone can get a clear view of it. Since then i was planning to prepare a project on it. But because of lack of knowledge about it, I was unable to do it.

When we meet our mentor last year to discuss the topic of our final year project. All of our group members also agreed with me to do the project on Handwritten Character Recognition.

Our mentor Prof. Piyal Sarkar also appreciated us and regularly motivated and guided. He gave us visualization and scope of this project in future which really worked a lot to complete this project.

Last month when I was doing training on Machine Learning, I came to know that it can be done using PyTesseract Library which will deduct text in different languages.

Chapter – 2

LITERATURE REVIEW

2.1 Introduction

Character recognition is better known as optical character recognition (OCR) since it deals with recognition of optically processed-characters rather than magnetically processed. Though the origin of character recognition can be found as early as 1870, it first appeared as an aid to visually handicapped, and the first successful attempt was made by the Russian Scientist Tyurin in 1900.

The modern version of OCR appeared in the middle of 1940s with the development of digital computers. The principal motivation for the development of OCR systems is the need to cope with the enormous flood of paper like bank cheques, commercial forms, government records, credit card imprints and mail sorting. OCR machines are commercially available since middle 1950s. Since then extensive research has been carried out and large number of technical papers, reports and books have been published on OCR. Research work also appeared in various other conferences such as British Conference on Pattern Recognition.

Presently, the methodologies in OCR have advanced to sophisticated techniques for recognition of a wide variety of complex handwritten characters in different language.

2.2 OCR classification

On the basis of the capabilities and complexity offline OCR can be classified as

1) Fixed font OCR
2) Multi font OCR
3) Omni font OCR
4) Handwritten OCR
5) Script Recognition

2.3 RESEARCH WORK IN CHARACTER RECOGNITION

Following are the few important papers related to research work in some recent years

1) Keiji Yamada have suggested a concept of locally connected and globally connected MLP ANN for HNR. They have also refined the original BP to evade a 'standstill' in learning.
2) Agui Takeshi used moment invariants and MLP trained by BP for recognition of handwritten Katakana in a frame.
3) Oliver has presented two stage .classification procedure for the recognition of Thai characters. The characters are classified into a set of groups and then characters in each group are classified. MLP ANN trained with BP is used in both the stages.
4) A Krzyzak has used modified BP for unconstrained HCR.

5) Cho Sung Bae has presented three sophisticated ANN classifiers to solve complex handwritten numeral recognition problem -Multiple MLP, Hidden Markov Model / MLP Hybrid Model and Adaptive Self Organization Map (SOM)

6) T.D.Vogi has used modified BP for recognition of handwritten digits and Japanese Kanji.

Chapter – 3

Definition of the problem with the modules and functionalities

3.1 What is Optical Character Recognition

As we read words on our computer screen, our eyes and brain are carrying out optical character recognition without we even noticing! Our eyes are recognizing the patterns of light and dark that make up the characters (letters, numbers, and things like punctuation marks) printed on the screen and our brain is using those to figure out what I'm trying to say (sometimes by reading individual characters but mostly by scanning entire words and whole groups of words at once).

Computers can do this too, but it's really hard work for them. The first problem is that a computer has no eyes, so if we want it to read something like the page of an old book, we have to present it with an image of that page generated with an optical scanner or a digital camera. The page we create this way is a graphic file (often in the form of a JPG) and, as far as a computer's concerned, there's no difference between it and a photograph of the Taj Mahal or any other graphic: it's a completely meaningless pattern of pixels (the colored dots or squares that make up any computer graphic image). In other words, the computer has a *picture* of the page rather than the text itself—it can't read the words on the page like we can, just like that. OCR is the process of turning a picture of text into text itself—in other words, producing something like a TXT or DOC file from a scanned JPG of a printed or handwritten page.

It involves the detection of text content on images and translation of the images to encoded text that the computer can easily understand. An image containing text is scanned and analyzed in order to identify the characters in it. Upon identification, the character is converted to machine-encoded text.
Text on an image is easily discernible and we are able to detect characters and read the text, but to a computer, it is all a series of dots.
The image is first scanned and the text and graphics elements are converted into a bitmap, which is essentially a matrix of black and white dots. The image is then pre-processed where the brightness and contrast are adjusted to enhance the accuracy of the process.
Then image is split into zones identifying the areas of interest such as where the images or text are and this helps in initiation of the extraction process. The areas containing text can now be broken down further into lines and words and characters and now the software is able to match the characters through comparison and various detection algorithms. The final result is the text in the image that we're given.
The process may not be 100% accurate and might need human intervention to correct some elements that were not scanned

correctly. Error correction can also be achieved using a dictionary or even Natural Language Processing (NLP).

3.2 Uses of OCR

Previously, digitization of documents was achieved by manually typing the text on the computer. Through OCR, this process is made easier as the document can be scanned, processed and the text extracted and stored in an editable form such as a word document.
Other uses of OCR include automation of data entry processes, detection, and recognition of car number plates.

3.3 What we'll Use

For this OCR project, we will use the Python-Tesseract, or simply PyTesseract, library which is a wrapper for Google's Tesseract-OCR Engine.
We chose this because it is completely open-source and being developed and maintained by the giant that is Google.
We will also use the Flask web framework to create our simple OCR server where we can take pictures via the webcam or upload photos for character recognition purposes.
We are also going to use Pipenv since it also handles the virtual-environment setup and requirements management.
Besides those, we'll also use the Pillow library which is a fork of the Python Imaging Library (PIL) to handle the opening and manipulation of images in many formats in Python.

3.4 Improving the quality of the output

OCR is not always 100% accurate and may need human intervention from time to time. There are a variety of reasons of not getting good quality output from Tesseract. It's because that, Pytesseract is basically good at recognising computerised fonts. Contrast between the text and the background is also the reason for the poor detection of handwritten text.

Chapter – 4

Software Design DFDs, UML

4.1 DATA FLOW DIAGRAM

4.1.1 What is DFD?

The DFD is also called as bubble chart. A data-flow diagram(DFD) is a graphical representation of the "flow" of data through an information system. DFD's can also be used for the visualization of data processing. Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

4.1.2 DFD Symbols

There are **three basic symbols** that are used to represent a data-flow diagram in this report.

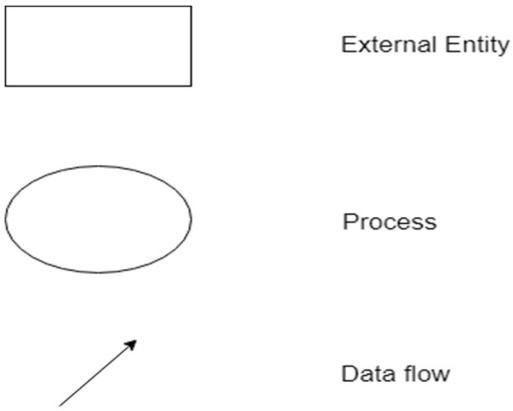


Fig.4.1.1

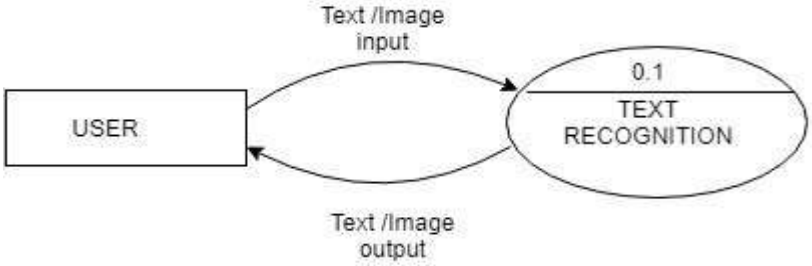
4.1.3 Why DFD?

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams. DFD has often been used due to the following reasons:

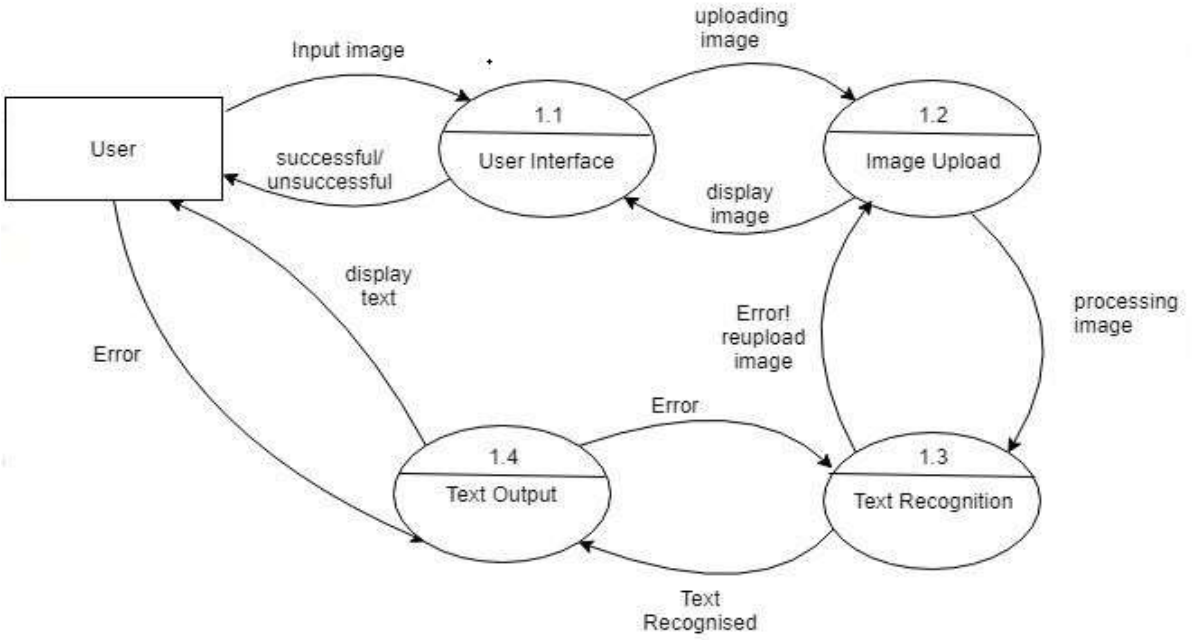
- Logical information flow of the system
- Determination of physical system construction requirements
- Simplicity of notation
- Establishment of manual and automated systems requirements

4.1.4 Handwritten Text Recognition DFD

The flow of data in our system can be describe in the form of dataflow diagram as follow



LEVEL 0



LEVEL 1

Fig. 4.1.2

In the above DFD we have explain the whole process in one level of DFD diagram as follow:-

Level 0

The user will interact with user interface of this software to perform the actions to extract the text from the hand written image. Where User will input the image and image will process and extracted text will display on user interface

Level 1

We have describe the whole backend process of this system regarding data flow.

Level 1.1

First the user will input the image after input of image a message will receive either a successful image upload or unsuccessful image.

Level 1.2

If the message will successful then user will upload the image by pressing upload button. After uploading the image will display on the user interface. A message will print "image is processed".

Level 1.3

In this level image will processed in the backend process using python code. Where set of different language dataset are present in pytesseract library.

Level 1.4

After image processing the individual character is recognise and converted in to string and send to the user interface.

4.2 Use case Diagram

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of **visually representing a system** along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

4.2.1 What is UML?

UML is an acronym that stands for **Unified Modeling Language**. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

UML was created as a result of the chaos revolving around software development and documentation. In the 1990s, there were several different ways to represent and document software systems. The need arose for a more unified way to visually represent those systems and as a result, in 1994-1996, the UML was developed by three software engineers working at Rational Software. It was later adopted as the standard in 1997 and has remained the standard ever since, receiving only a few updates.

4.2.2 What is the use of UML?

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficacy.

UML itself finds different uses in software development and business process documentation:

Sketch

UML diagrams, in this case, are used to communicate different aspects and characteristics of a system. However, this is only a top-level view of the system and will most probably not include all the necessary details to execute the project until the very end.

- **Forward Design**

  The design of the sketch is done before coding the application. This is done to get a better view of the system or workflow that you are trying to create. Many design issues or flaws can be revealed, thus improving the overall project health and well-being.

- **Backward Design**

  After writing the code, the UML diagrams are drawn as a form of documentation for the different activities, roles, actors, and workflows.

Blueprint

In such a case, the UML diagram serves as a complete design that requires solely the actual implementation of the system or software. Often, this is done by using CASE tools (Computer Aided Software Engineering Tools). The main drawback of using CASE tools is that they require a certain level of expertise, user training as well as management and staff commitment.

Pseudo Programming Language

UML is not a stand-alone programming language like Java, C++ or Python, however, with the right tools, it can turn into a pseudo programming language. In order to achieve this, the whole system needs to be documented in different UML diagrams and, by using the right software, the diagrams can be directly translated into code. This method can only be beneficial if the time it takes to draw the diagrams would take less time than writing the actual code.

Despite UML having been created for modeling software systems, it has found several adoptions in business fields or non-software systems.

The use case view models functionality of the system as perceived by outside uses. A use case is
a coherent unit of functionality expressed as a transaction among actors and the sys-tem.

4.2.3 UML of Handwritten Text Recognition

In the above UML Diagram "User" and "System" act as two actors.The steps taken are as follows:-

1.User(Actor 1)

a) The image is uploaded by the user.
b) User may change or cancel the picture if want to do so.

2.System(Actor 2)

After uploading the image by user now the action is taken by System.

a) Intialization

Intialization of the image is done using Intialization process in which it is checked whether the image is uplaoded or not.

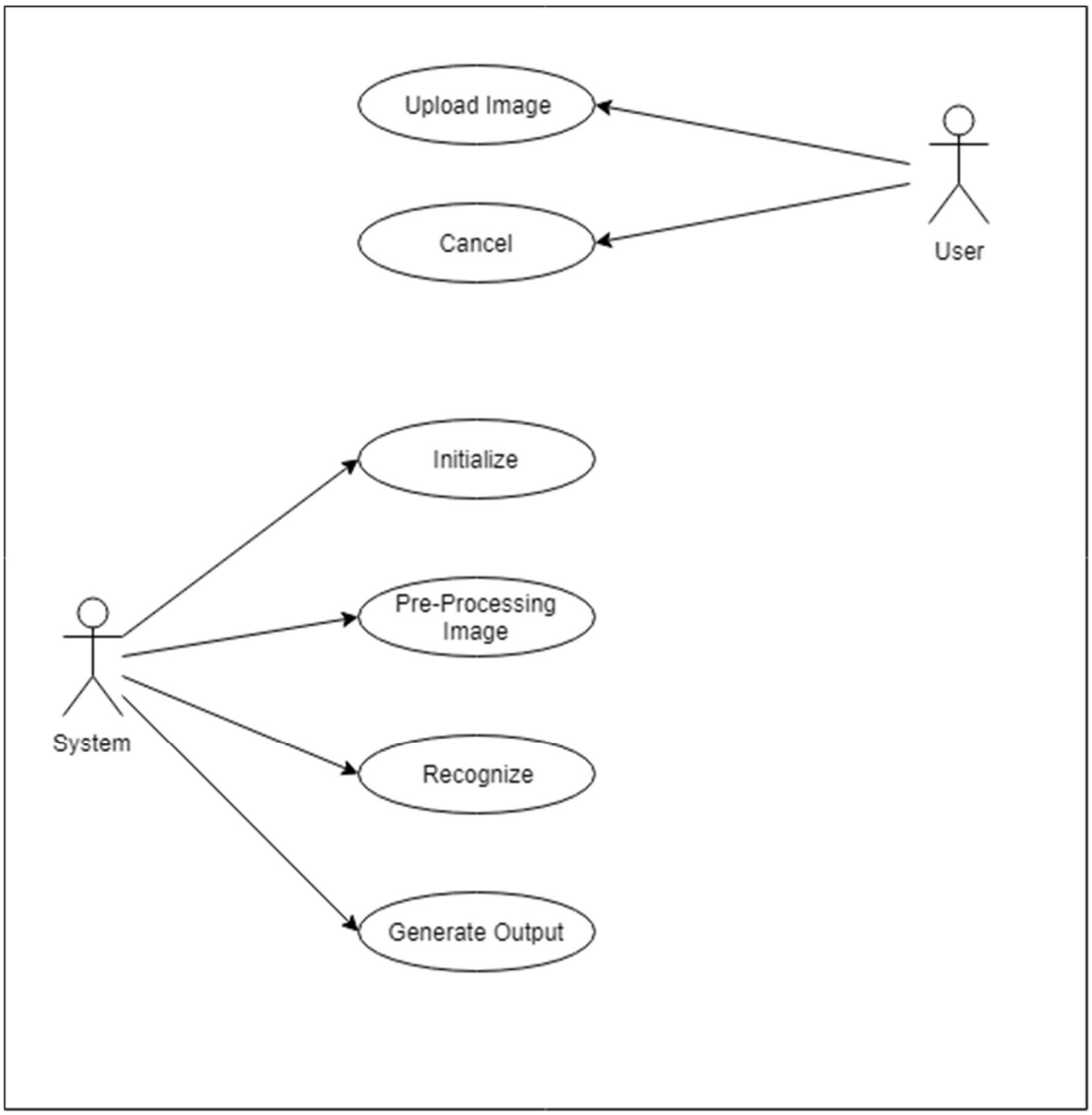


Fig.4.2.1

b) Pre-processing

Now the System executes the pre-processing of image. Here It is checked whether the size of image taken supports the format ,sacnned image is in png,jpg or jpeg or not.Whether the quality of image taken is proper or not.

c) Recognition

After pre-processing the system execute the recognization process. In this backend process the text in the image is matched by the datasets installed in the pytesseract library.

d) Output

The extraxted text is converted into strong and the output is genearted. A user friendly GUI page is designed on which extracted text is printed .

Chapter-5

Software and Hardware requirements

5.1 Software Requirements :

1. Python 3.7.3
2. PyTesseract Library
3. Pipnev
4. Pillow Library
5. Visual Studio Code Editor

5.1.1 Python 3.7.3

The backend process is in Python. So we need python installed in our system. We have used latest version of python ie python 3.7.3 available at that time.

5.1.2. PyTesseract Library

we will use the Python-Tesseract, or simply *PyTesseract*, library which is a wrapper for Google's Tesseract-OCR Engine.

We chose this because it is completely open-source and being developed and maintained by the giant that is Google.

5.1.3 Pipnev

We have used Pipenv since it handles the virtual-environment setup and requirements management.

5.1.4. Pillow Library

Pillow library is a fork of the *Python Imaging Library* (PIL) to handle the opening and manipulation of images in many formats in Python. In our project we can upload jpeg, jpg and png files to recognise text from it.

5.1.5 Visual Studio Code Editor

We need a code editor to write and edit python, html, css codes. So we have used Microsoft Visual Studio code editor. Because it is good for debugging our codes.
Other code editor can also be used viz. Atom, notepad, subline text etc.

5.2 Hardware Requirements :

1. Windows OS
2. Mobile Camera Scanner Or simply Scanner

5.2.1 Windows OS

To run our project hand written text recognition , we need a hardware platform where we can install and set environment variable. So we an Operating System and we have used windows 10 OS. Other OS can also be used like Ubuntu, mac, Florida etc. But for that we will have different setup.

5.2.2 Mobile Camera Scanner Or simply Scanner

We need a scanner to scan our images and upload it to our UI to recognise text. Any mobile scanner can be used for that. But we have used CamScanner for simplicity.

Chapter - 6

Code Templates

6.1 Installation and environment setups:

Start by installing *Pipenv* using the following command via Pip .

```
$ pip install pipenv
```

Create the project directory and initiate the project by running the following command:

```
$ mkdir ocr_server && cd ocr_server && pipenv install –three
```

We can now activate our virtual environment and start installing our dependencies:

```
$ pipenv shell
$ pipenv install pytesseract Pillow
```

In case you'll not be using Pipenv, you can always use the Pip and Virtual Environment approach. Follow the official documentation to help you get started with Pip and Virtual Environment:

Note: In that case, instead of `pipenv install Pillow`, the command will be `pip install Pillow`.

6.2 Implementation

We are going to implement this project in 2 phases. In the first, we'll create the script, and in the next we'll build a Flask application to act as an interface

6.2.1 OCR Script :

With the setup complete, we can now create a simple function that takes an image and returns the text detected in the image - this will be the core of our project:

```
try:
   from PIL import
Image except
ImportError:
   import Image
import pytesseract
def
ocr_core(filename):
"""
This function will handle the core OCR processing of
images. """
text = pytesseract.image_to_string(Image.open(filename))    # We'll
use Pillow's Image class to open the image and pytesseract to detect
the string in the image
   return text
```

The function is quite straightforward, in the first 5 lines we import Imagefrom the Pillow library and our PyTesseract library.

We then create and ocr_core function that takes in a file name and returns the text contained in the image.

Let's integrate this script into a Flask application first, to make it easier to upload images and perform character recognition operation.

6.2.2 Flask Web Interface

Our script can be used via the command line, but a Flask application would make it more user-friendly and versatile. For instance, we can upload photos via the website and get the extracted text displayed on the website or we can capture photos via the web camera and perform character recognition on them.

If you are unfamiliar with the Flask framework, this is a good tutorial to get you up to speed and going.

Let's start by installing the Flask package:

```
$ pipenv install Flask
```

Now, let's define a basic route:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home_page():
    return "Hello World!"

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

Save the file and run:

```
$ python3 app.py
```

If you open your browser and head on to `127.0.0.1:5000` or `localhost:5000` you should see "Hello World!" on the page. This means our Flask app is ready for the next steps.

We'll now create a `templates` folder to host our HTML files. Let's go ahead and create a simple `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    Hello World.
  </body>
</html>
```

Let us also tweak our app.py to render our new template:

```
from flask import Flask, render_template
app = Flask(__name__)

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

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

Notice we have now imported render_template and used it to render the HTML file. If you restart your Flask app, you should still see "Hello World!" on the home page.

That's enough on the Flask crash course, let us now integrate our OCR script on the web application.

First, we'll add functionality to upload images to our Flask app and pass them to the ocr_core function that we wrote above. We will then render the image beside the extracted text on our web app as a result:

```
import os
from flask import Flask, render_template, request

# import our OCR function
from ocr_core import ocr_core

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

```
# allow files of a specific type

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])
app = Flask(__name__)

# function to check the file extension

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# route and function to handle the home page

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

# route and function to handle the upload page

@app.route('/upload', 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',

                                   msg='Successfully processed',
                                   extracted_text=extracted_text,
                                   img_src=UPLOAD_FOLDER + file.filename)

    elif request.method == 'GET':

        return render_template('upload.html')

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

As we can see in our upload_page() function, we will receive the image via *POST* and render the upload HTML if the request is *GET*.

We check whether the user has really uploaded a file and use the function allowed_file() to check if the file is of an acceptable type.

Upon verifying that the image is of the required type, we then pass it to the character recognition script we created earlier.

The function detects the text in the image and returns it. Finally, as a response to the image upload, we render the detected text alongside the image for the user to see the results.

The upload.html file will handle the posting of the image and rendering of the result by the help of the Jinja templating engine, which ships with Flask by default.

```
<!DOCTYPE html>
<html>
<head>
    <title>Upload Image</title>
    <style>

       #header{
               float:left;
               margin-left:100px;

        }
        #msg{
             float:right;
             margin-right:400px;

        }

       .upload{
            float:left;
            margin-left:100px;
            border-radius: 25px;
            background-color:#FFFFFF;
            border: 1px solid green ;
            height:350px;
            width:35%;

       }
       .prediction{
          float:left;
          margin-left:100px;
          border-radius: 25px;
          background-color:#FFFFFF;
          border: 1px solid green ;
          height:350px;
          width:35%;
     }
```

```
.second{
        margin-left: 10px;
        padding-left: 5px;
     }

</style>
</head>
<body bgcolor="#AEB6BF">
    <center>
    <h1>HAND WRITTEN TEXT RECOGNITION</h1>


    <form method=post enctype=multipart/form-data>
         <h1>Upload Image</h1>
    <p>
            <input type=file name=file>
            <input type=submit value=Upload>
    </form>
    </br></br>
    {% if msg %}
    <h1>{{ msg }}</h1>
    {% endif %}
    </br>
    <div>
    </center>
    <div class = "box">
                <div class="upload">
                    <center><h1>Your Image</h1></center>
                        <div class="first">
                            {% if img_src %}
                            <img src="{{ img_src }}" style = "height= 99%; width=99%;
object-fit:contain;">
                        {% endif %}
                        </div>
                </div>

                <div class="prediction">
                    <center><h1>Recognised Text</h1></center>


  {% if extracted_text %}
                        <div class="second">
                            <b style="margin-left:10px;margin-right:10px;padding-top: 5px;
padding-right: 5px; padding-bottom: 5px;padding-left: 5px;">
                                {{ extracted_text }}
                            </b>
                        </div>

                        {% else %}
                        The extracted text will be displayed here
                        {% endif %}
                </div>
            </div>

    </div>

</body>
</html>
```

Jinja templating allows us to display text in specific scenarios through the `{% if %} {% endif %}` tags. We can also pass messages from our Flask app to be displayed on the webpage within the `{{ }}` tags. We use a form to upload the image to our Flask app.

Chapter – 7

Testing

Steps needed to follow to run our project :

1. To run and test our project, we need to start flask for which we have to run flask\Scripts\activate in cmd after being in our project directory.

2. We will run our app.py file

3. Then we have to go to any browser and run on http://localhost:5000/upload

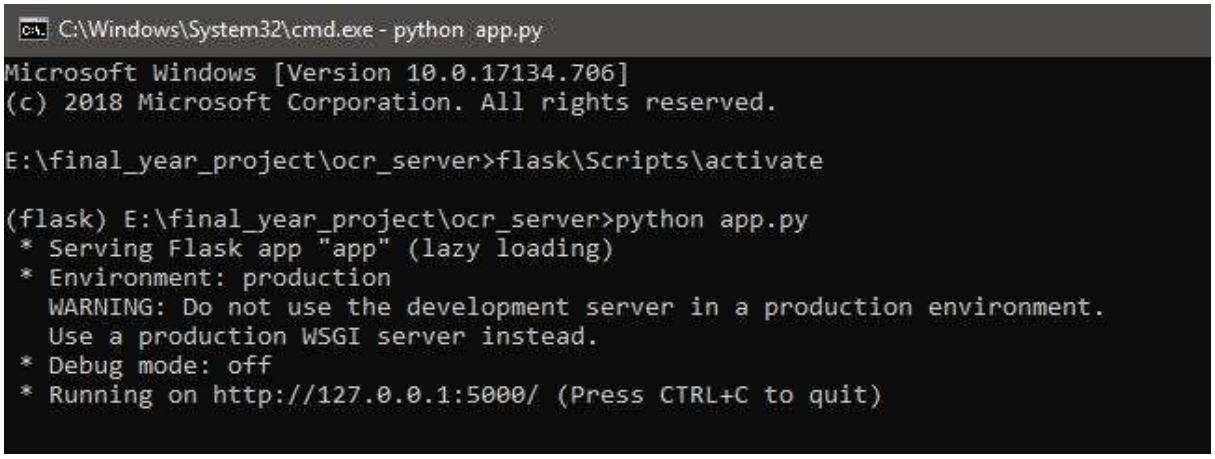


Fig. 7.1

Test Case 1 :

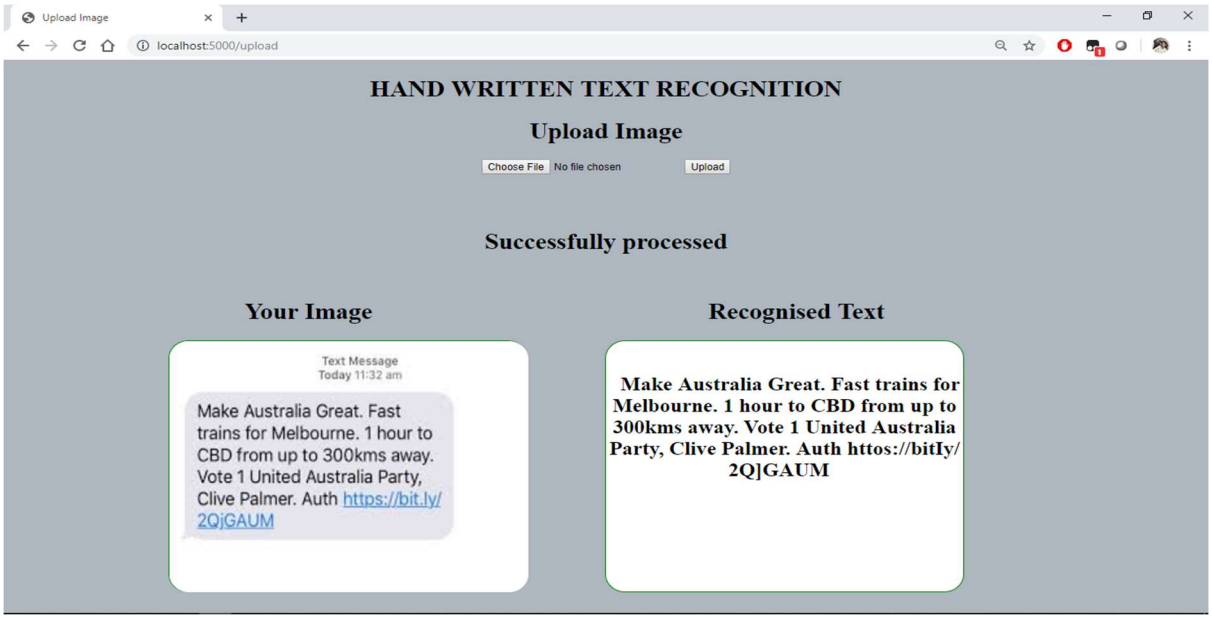


Fig. 7.2

The output or recognised text is is almost 100% accurate.

Test Case 2 :

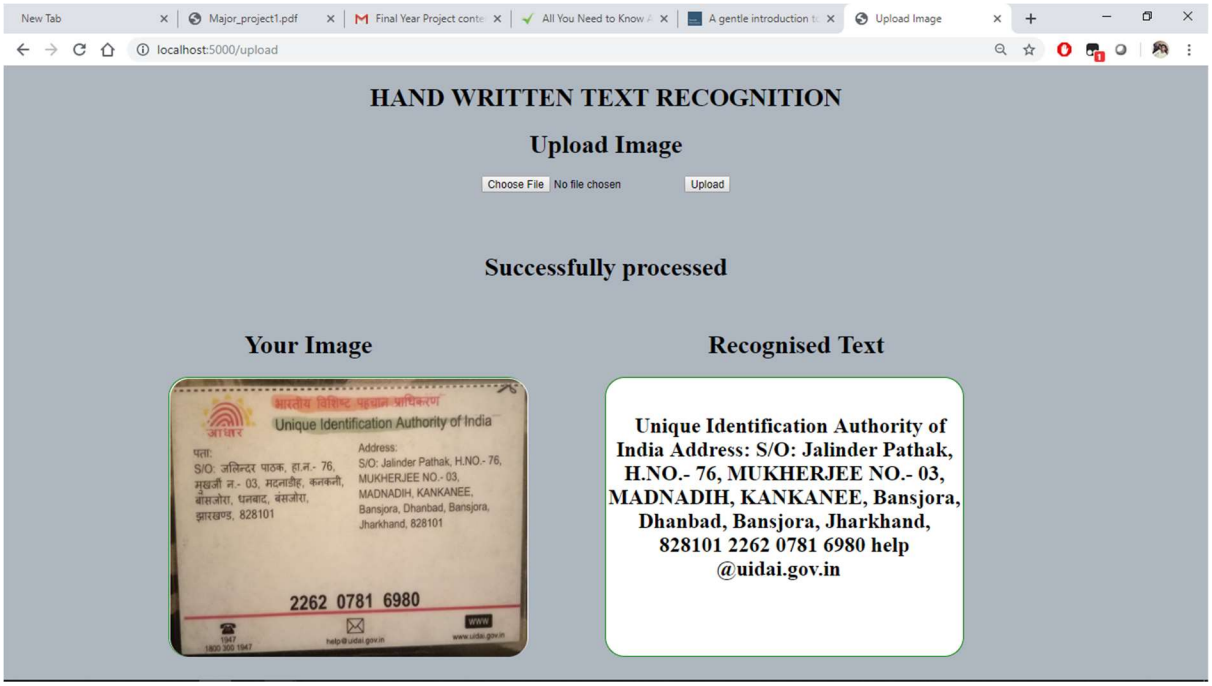


Fig. 7.3

Test Case 3 :

In this test case we have taken a handwritten text and checked it.

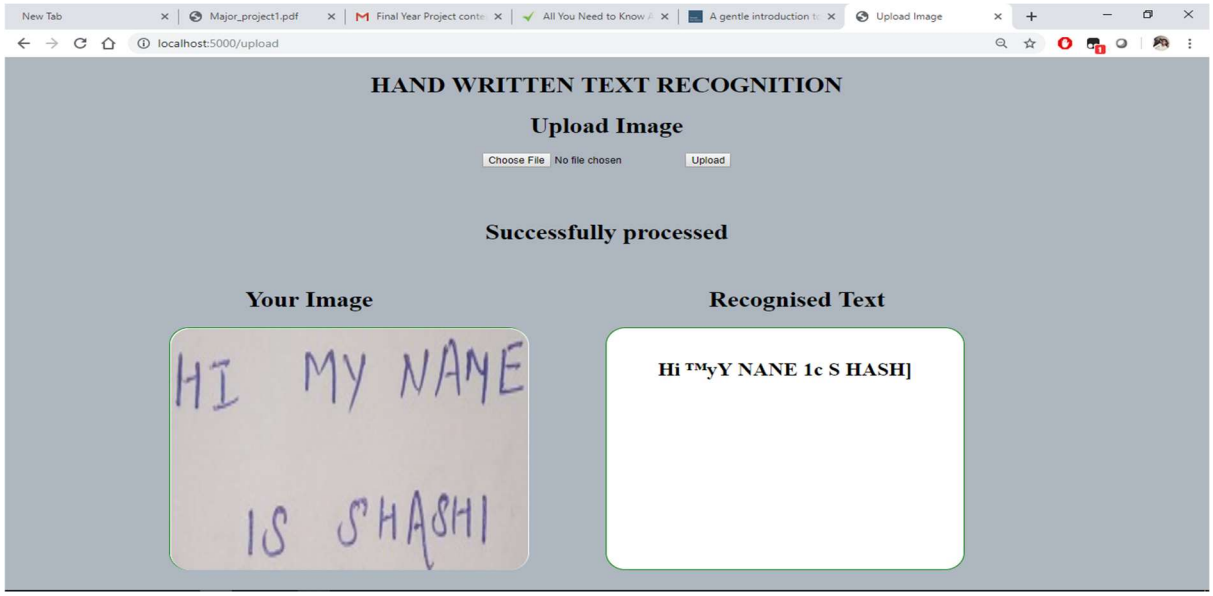


Fig. 7.4

The output of this image is not 100% accurate but some of the words have been recognised correct. Probably this is because of poor handwriting.

Test Case 4 :

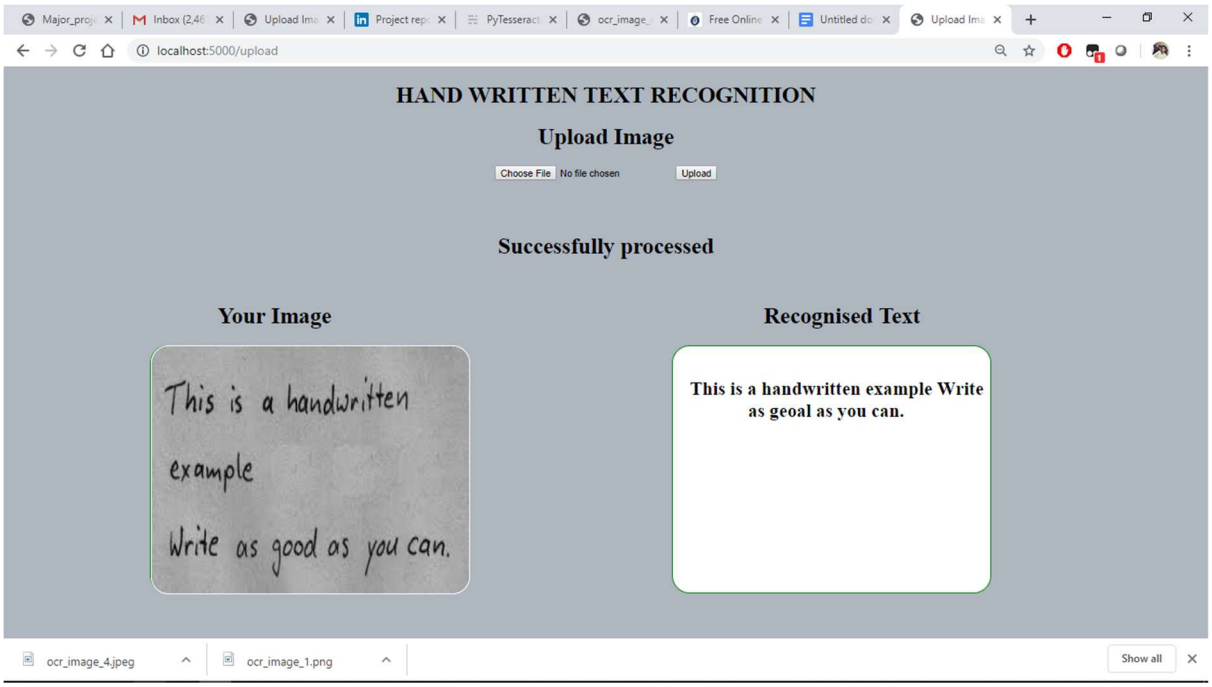


Fig. 7.5

The character recognition on this image is much better than the than the previous handwritten image. As we can see the lines in this image are thicker and there's better contrast between the text and the background and this could be the reason for the poor detection of previous handwritten text.

This is an area to explore further, we can get handwritten notes from friends or colleagues and see how well the script will be able to detect characters.

Conclusion

We have been able to scan images and extract text through Tesseract and the Python-Tesseract library. This is Optical Character Recognition and it can be of great use in many situations. We have built a scanner that takes an image then converts it in a string and returns the text contained in the image and integrated it into a Flask application as the interface. This allows us to expose the functionality in a more familiar medium and in a way that can serve multiple people simultaneously.

It is important to note that accuracy can be improved by testing an image of high dpi value (atleast 300dpi) and good resolution i.e HD resolution. This Project is tremendous work of teammates. Although the accuracy of handwritten text is not 100% probably because of poor contrast ratio of background and text. Also the Pytesseract library is good at recognising computerised fonts and languages. So it is hard to get 100% accuracy of hand written text.

In this project we have only focused on recognising text of English language. But it can be improved to be capable of recognising other languages if we will have installed PyTesseract library of that language. We have not put a lot of effort on developing UI of it since a lot work and time have been consumed in recognising accurate text. Also the current status of the project can only recognise text from jpeg, jpg, and png files only. But this can also be customised for detecting text from pdf files.

References

- https://stackabuse.com/pytesseract-simple-python-optical-character-recognition/
- https://www.youtube.com/watch?v=uU1LHp-wFDQ
- https://www.youtube.com/watch?v=aZsZrkIgan0
- https://www.kaggle.com/c/digit-recognizer/data
- https://scikit-learn.org/stable/
- https://www.geeksforgeeks.org/project-idea-character-recognition-from-image/
- https://www.codeproject.com/Articles/476142/Optical-Character-Recognition