

# **EXTRACT AND ORGANIZE INFORMATION IN IMAGES WITH AI USING IBM SERVICES**

## **Mini Project Report**

Submitted By (**BATCH NO: CSE\_08**)

<b>CHEEKATI SAIRACHANA</b>	<b>18UK1A05A6</b>
<b>KATKOJU SANKEERTHI</b>	<b>18UK1A05E4</b>
<b>RAMARAPU YAMINI</b>	<b>18UK1A05A2</b>
<b>CHILPURI SHIVATEJA</b>	<b>18UK1A05C7</b>

# INDEX

<b>TITLE</b>	<b>PAGE NO</b>
<b>CHAPTER 1: INTRODUCTION</b>	<b>4</b>
<b>1.1 OVERVIEW</b>	<b>4</b>
<b>1.2 PURPOSE</b>	<b>4</b>
<b>CHAPTER 2: LITERATURE SURVEY</b>	<b>5</b>
<b>2.1 EXISTING PROBLEM</b>	<b>5</b>
<b>2.2 PROPOSED SOLUTION</b>	<b>5</b>
<b>CHAPTER 3: THEORTICAL ANALYSIS</b>	<b>6</b>
<b>3.1 BLOCK DIAGRAM</b>	
<b>3.2 HARDWARE/SOFTWARE DESIGNING</b>	
<b>CHAPTER 4: PROJECT FLOW</b>	<b>7</b>
<b>CHAPTER 5: RESULTS</b>	<b>8</b>
<b>CHAPTER 6: APLPLICATIONS</b>	<b>9</b>
<b>CHAPTER 7: CONCLUSION</b>	<b>9</b>
<b>CHAPTER 8: FUTURE SCOPE</b>	<b>9</b>
<b>CHAPTER 9: BIBILOGRAPHY</b>	<b>10</b>
<b>CHAPTER 10: APPENDIX</b>	<b>10</b>

## **ABSTRACT**

Humans are bound to make errors-some time or the other – especially while performing mundane boring tasks like digitization or security, Continuously Many times we are unable to perceive certain digits due to various factors-motion, lack digit clarity, illumination and so on. It is these problems which have to lead us to delve into this topic

We as a team going to tackle with the advent of OCR techniques, much time has been saved by automatically extracting the text out of a digital image of any invoice or a document. Currently, this is where most organizations that use OCR for any form of automation are Digital copies of invoices or documents are obtained by scanning or taking pictures. The text is extracted from these documents is entered into a template-based data entry software.

The project aims at creating an application form where the user can upload a pdf document/Image containing text, the document is analysed by an Optical character recognition (OCR) to extract text from it. The extracted text is again saved in a text document in the local drive.

# **1. INTRODUCTION**

## **1.1 Overview**

Literally, OCR stands for Optical Character Recognition. It is a widespread technology to recognize text inside images, such as scanned documents and photos. OCR technology is used to convert virtually any kind of image containing written text (typed, handwritten, or printed) into machine-readable text data.

OCR Technology became popular in the early 1990s while attempting to digitize historic newspapers. Since the technology has undergone several improvements. Nowadays solutions deliver near to perfect OCR accuracy. Advanced methods like Zonal OCR are used to automate complex document-based workflows.

OCR is often used as a “hidden” technology, powering many well-known systems and services in our daily life. Less known, but as important, use cases for OCR technology include data entry automation, indexing documents for search engines, automatic number plate recognition, as well as assisting blind and visually impaired persons.

## **1.2 Purpose**

Optimal character recognition system based on a grid infrastructure is to perform image analysis, document processing of electronic document formats converted from paper formats.

The primary objective is to speed up the process of character recognition in document processing.

## **2. LITERATURE SURVEY**

### **2.1 EXISTING PROBLEM**

In the running world there is a growing demand for the users to convert the printed documents in to electronic documents for maintaining the security of their data. Hence the basic OCR system was invented to convert the data available on papers in to computer process able documents,

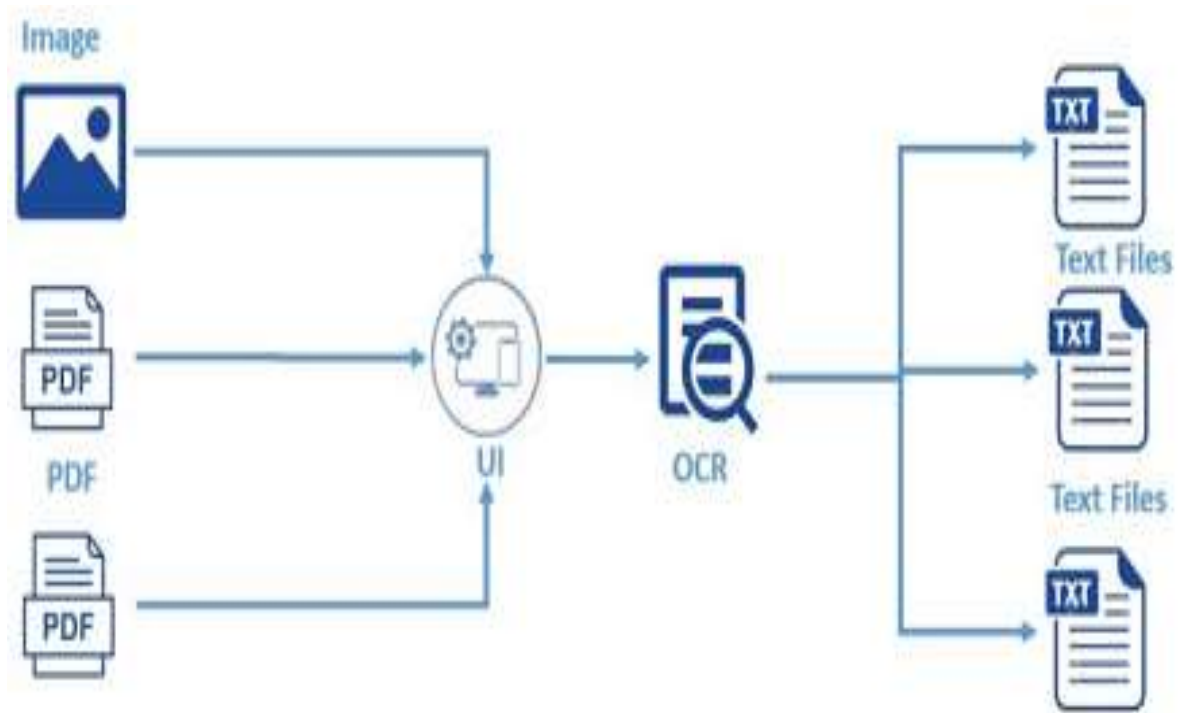
So that the documents can be editable and reusable

### **2.2 PROPOSED SYSYTEM**

Our proposed system is OCR on a grid infrastructure which is a character recognition system that supports recognition of the characters of multiple languages. This feature is what we call grid infrastructure which eliminates the problem of heterogeneous character recognition and supports multiple functionalities to be performed on the document. The multiple functionalities include editing and searching too where as the existing system supports only editing of the document. In this context, grid infrastructure means the infrastructure that supports group of specific set of languages, Thus OCR on a grid infrastructure is multi-lingual.

### 3. THEORETICAL ANALYSIS

#### 3.1 BLOCK DIAGRAM



#### 3.2 HARDWARE/SOFTWARE DESIGNING

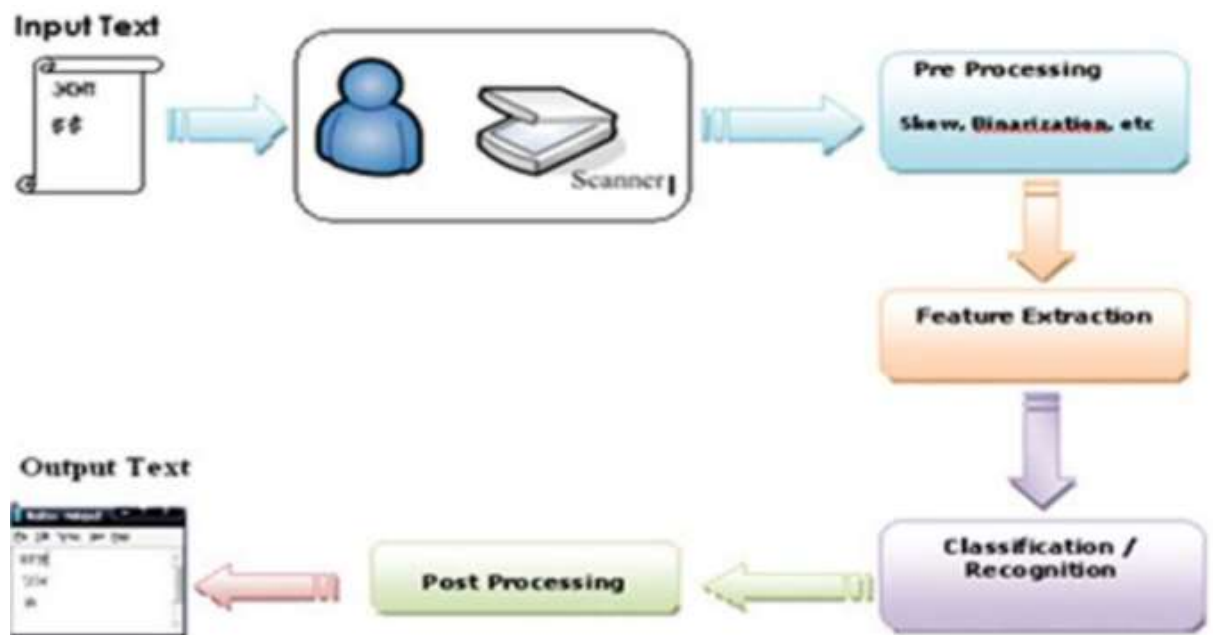
To complete this project you should have the following software and packages.

- Anaconda Navigator  
Jupyter notebook
- Command prompt
- pytesseract
- pdf2image

## 4. PROJECT FLOW

To accomplish this, we have to complete all the activities and tasks listed below

- Upload a pdf document
- Convert pdf document to Image
- Extract the text from Image
- Store the extracted text in the document




- Data Preprocessing
  - Analyze the data
- Model Building
  - Import the Model building Libraries
  - Initializing the model
- Application Building
  - Create an HTML file
  - Build Python Code

## 5. RESULTS

Smart OCR


Smart OCR :

Optical character recognition or optical character reader is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image.



Upload PDF Here

Choose... Choose File No file chosen



Converted file :  
C:\Users\Rachana\Desktop\min\ocr - flask\outputs\output56116.txt



## **6. APPLICATIONS**

- When you scan a document that has text or numeric data on it, you are able to read and understand what is written in the scanned image. However, to a computer, the resulting image file is just as meaningless an assortment of pixels as a landscape photo. In order to transform this information into an editable format that you can search through, copy, and modify without retyping it manually, you will need the an Optical Character Recognition (OCR) software.
- There is a wide variety of OCR software available. While they all share the ability to convert images of machine printed (not handwritten) text or numbers into an editable format, the various software often have different features, accuracy, prices, and language options.

## **7.CONCLUSION**

- OCR results depend on the input data quality. A clean segmentation of the text and no noise in the background gives better results. In the real world, this is not always possible, so we need to apply multiple pre-processing techniques for OCR to give better results.

## **8. FUTURE SCOPE**

The Optical character Recognition software can be enhanced in the future in different kinds of ways such as:

- Training and recognition speeds can be increased greater and greater by mistake is more user-friendly. Many applications exist where it would be desirable to read handwritten entries. Reading handwriting is a very difficult task considering the diversities that exist in ordinary penmanship. However, progress is being made.

## 9. BIBLIOGRAHY

- <https://www.geeksforgeeks.org/>
- <https://www.w3schools.com/>
- <https://stackoverflow.com/>

## 10. APPENDIX

### base.html

```
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Smart OCR</title>
  <link href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet">
  <script src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper.min.js"></script>
  <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
  <link href="{{ url_for('static', filename='css/main.css') }}" rel="stylesheet">
</style>
.bg-dark {
    background-color: #42678c!important;
}
#result {
    color: #0a1c4ed1;
}
</style>
</head>

<body>
  <nav class="navbar navbar-dark bg-dark">
    <div class="container">
```

```

        <a class="navbar-brand" href="#">Smart OCR</a>
    </div>
</nav>
<div class="container">
    <div id="content" style="margin-top:2em">
        <div class="container">
            <div class="row">
                <div class="col-sm-6 bd" >
                    <h3>Smart OCR : </h3>
                    <br>
                    <p>Optical character recognition or optical character reader is the
electronic or mechanical conversion of images of typed, handwritten or printed text into
machine-encoded text, whether from a scanned document, a photo of a document, a scene-
photo or from subtitle text superimposed on an image.</p>
                    
                </div>
                <div class="col-sm-6">
                    <div>
                        <h4>Upload PDF Here</h4>
                        <form action = "http://localhost:5000/" id="upload-file" method="post"
enctype="multipart/form-data">
                            <label for="imageUpload" class="upload-label">
                                Choose...
                            </label>
                            <input type="file" name="image" id="imageUpload" accept=".png,
.jpg, .jpeg, .pdf">
                        </form>

                        <div class="image-section" style="display:none;">
                            <div class="img-preview">
                                <div id="imagePreview">
                                    </div>
                                </div>
                                <div>
                                    <button type="button" class="btn btn-info btn-lg " id="btn-
predict">Convert</button>
                                </div>
                            </div>
                        </div>
                    
```

```

        <div class="loader" style="display:none;"></div>

        <h3>
            <span id="result"> </span>
        </h3>
        <a href="output/output.txt" download="output">

    </a>

    </div>
    </div>

    </div>
    </div>
    </div>
</div>
</body>

<footer>
    <script src="{{ url_for('static', filename='js/main.js') }}" type="text/javascript"></script>
</footer>

</html>

```

### **main.css**

```

.img-preview {
    width: 256px;
    height: 256px;
    position: relative;
    border: 5px solid #F8F8F8;
    box-shadow: 0px 2px 4px 0px rgba(0, 0, 0, 0.1);
    margin-top: 1em;
    margin-bottom: 1em;
}

.img-preview>div {
    width: 100%;
    height: 100%;

```

```

    background-size: 256px 256px;
    background-repeat: no-repeat;
    background-position: center;
}

input[type="file"] {
    display: none;
}

.upload-label{
    display: inline-block;
    padding: 12px 30px;
    background: #39D2B4;
    color: #fff;
    font-size: 1em;
    transition: all .4s;
    cursor: pointer;
}

.upload-label:hover{
    background: #34495E;
    color: #39D2B4;
}

.loader {
    border: 8px solid #f3f3f3; /* Light grey */
    border-top: 8px solid #3498db; /* Blue */
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

```

**ocr.py**

```

import numpy as np
import cv2
from PIL import Image
import pytesseract
import sys
from pdf2image import convert_from_path
import sys
import os
import random

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract.exe'
pages = convert_from_path('sample.pdf', 500)
# Counter to store images of each page of PDF to image
image_counter = 1

    # Iterate through all the pages stored above
for page in pages:
    # Declaring filename for each page of PDF as JPG
    # For each page, filename will be:
    # PDF page 1 -> page_1.jpg
    # PDF page 2 -> page_2.jpg
    # PDF page 3 -> page_3.jpg
    # PDF page n -> page_n.jpg
    filename = "page_"+str(image_counter)+".jpg"
    # Save the image of the page in system
    page.save(filename, 'JPEG')
    # Increment the counter to update filename
    image_counter = image_counter + 1

'''
Part #2 - Recognizing text from the images using OCR
'''

# Variable to get count of total number of pages
filelimit = image_counter-1

basepath = os.path.dirname(__file__)
file_path2 = os.path.join(

```

```

        basepath, 'outputs', "output"+str(random.randint(1,
                                                    100000))+".txt")

f = open(file_path2, "a")

for i in range(1, filelimit + 1):

    filename = "page_"+str(i)+".jpg"

    text = str(((pytesseract.image_to_string(Image.open(
        filename))))))

    f.write(text)
    print(" text extracted is saved in ouput folder")

f.close()

```

### **hello.py**

```

from __future__ import division, print_function
from flask import Flask,request, render_template
#from werkzeug import secure_filename
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
import numpy as np
import cv2
from PIL import Image
import pytesseract
import sys
from pdf2image import convert_from_path
import os
pytesseract.pytesseract.tesseract_cmd = r'C:\Users\HP\AppData\Local\Tesseract-
OCR\tesseract.exe'
import sys
import os.path
import glob
import random

app = Flask(__name__, static_url_path='')

```

```

@app.route('/', methods=['GET'])
def index():
    return render_template('base.html')

@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']

        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        PDF_file = file_path
        # Store all the pages of the PDF in a variable
        pages = convert_from_path(PDF_file, 500)

        # Counter to store images of each page of PDF to image
        image_counter = 1

        # Iterate through all the pages stored above
        for page in pages:

            # Declaring filename for each page of PDF as JPG
            # For each page, filename will be:
            # PDF page 1 -> page_1.jpg
            # PDF page 2 -> page_2.jpg
            # PDF page 3 -> page_3.jpg
            # ....
            # PDF page n -> page_n.jpg
            filename = "page_"+str(image_counter)+".jpg"

            # Save the image of the page in system
            page.save(filename, 'JPEG')

```



```

        # Increment the counter to update filename
        image_counter = image_counter + 1

'''
Part #2 - Recognizing text from the images using OCR
'''

# Variable to get count of total number of pages
filelimit = image_counter-1

# Creating a text file to write the output
basepath = os.path.dirname(__file__)
file_path2 = os.path.join(
    basepath, 'outputs', "output"+str(random.randint(1, 100000))+".txt")

# Open the file in append mode so that
# All contents of all images are added to the same file
f = open(file_path2, "a")

# Iterate from 1 to total number of pages
for i in range(1, filelimit + 1):

    # Set filename to recognize text from
    # Again, these files will be:
    # page_1.jpg
    # page_2.jpg
    # ....
    # page_n.jpg
    filename = "page_"+str(i)+".jpg"

    # Recognize the text as string in image using pytesseract
    text = str(((pytesseract.image_to_string(Image.open(filename)))))

    # The recognized text is stored in variable text
    # Any string processing may be applied on text
    # Here, basic formatting has been done:
    # In many PDFs, at line ending, if a word can't
    # be written fully, a 'hyphen' is added.

```

```

# The rest of the word is written in the next line
# Eg: This is a sample text this word here GeeksF-
# orGeeks is half on first line, remaining on next.
# To remove this, we replace every '\n' to ".
text = text.replace('\n', ")

# Finally, write the processed text to the file.
f.write(text)

# Close the file after writing all the text.
f.close()
return file_path2

if __name__ == '__main__':
    port = int(os.getenv('PORT', 8000))
    #app.run(host='0.0.0.0', port=port, debug=True)
    http_server = WSGIServer(('0.0.0.0', port), app)
    http_server.serve_forever()
    app.run(debug=True)

```