Date :- 28 December 2024

A Weekly Report

Mr. Viraj Tank

[EC034]

[21ECUBG069]
B.Tech. SEM. VIII

In Partial Fulfillment of Requirement of Bachelor of Technology

Degree of Electronics & Communication Course

Submitted To

Faculty Supervisor

Dr. Narendra V. Chauhan

Department of Electronics & Communication Engineering

Faculty of Technology,

Dharmsinh Desai University, Nadiad-387001.

(March 2025)

Faculty's Sign **Mentor's Sign**

_____ _____

Task 3 :- Learning How Text Recognition Engine Works

Introduction :-

Text recognition is like teaching a computer to read words in pictures. Imagine you have a photo of a paper with words on it. A text recognition engine looks at the picture, figures out the words, and turns them into text that a computer and a human can understand.

In this chapter, we will learn how text recognition works, step by step, so we can understand how the computer reads and understands the words from pictures.

Definition's :-

1) **Model** :- A model is like a brain for a computer. It helps the computer learn how to do things, like reading text in pictures. When we teach a model, we show it many examples of images with text and tell it what the text says. The model then learns to recognize those patterns, so later, when it sees a new image, it can guess what the text is without being told.
2) **Engine** :- An engine is like a tool or machine that does the actual work of recognizing the text in an image. It uses the model to figure out the letters and words. Think of it like a car engine: the car doesn't move by itself, but with the engine working, the car goes forward. In the same way, the engine helps the model understand the text.
3) **OCR** :- OCR stands for Optical Character Recognition. It is the name of the whole process of turning text from pictures into text the computer can read. It's like a magic trick where the computer can see words in photos or scans and make them readable.

Difference between Model and Engine :-

	Model	Engine
Scope	Performs a single task.	Combines multiple tasks.
Input/Output	Inputs data; outputs predictions or results.	Takes raw input, processes it, and outputs usable results.
Role	A component of a larger system.	The complete system itself.
Example	LSTM model for recognizing characters.	Tesseract OCR for end-to-end text recognition.

Steps to create own OCR system :-

1) **Data Collection**

The first step is to gather the images that contain the text you want to recognize. These could be photos of documents, receipts, books, or handwritten notes.

2) **Annotation**

We gather pictures of text and mark where the words are. This helps the computer learn where the words are in the pictures.

3) **Preprocess the Images**

Image preprocessing helps in enhancing the input images to improve text detection and recognition accuracy. Example Grayscale Conversion, Thresholding, Noise Removal and etc.

4) **Text Detection**

Before recognizing text, you need to detect where the text is located in the image. Use object detection models to detect text as individual regions (bounding boxes).

5) **Text Recognition Model**

After detecting text regions, you need a model to recognize the characters or words inside the bounding boxes. This is the core of your OCR system. For this Various Deep Learning Models are used like CNN (Convolutional Neural Networks), RNN(Recurrent Neural Networks), LSTM(Long Short-Term Memory networks). This is like teaching the computer to read the words inside the boxes by showing it lots of examples

6) **Train Your Model**

We train the computer to get better at reading by showing it many pictures and correcting its mistakes.

7) **Post-Processing**

After the computer reads the words, we fix any mistakes it made, like correcting spelling.

8) **Optimization**

We check how well the computer is reading and make it even better by fixing problems.

9) **Deployment**

We make the computer ready to read words from new pictures, like putting it into an app that can read text for you.

Challenges of Creating Your Own OCR Model :-

- **Collecting Data**: You need lots of pictures with text, and each one has to be labelled correctly, which takes a lot of time.
- **Training the Model**: Teaching the model to read text takes powerful computers and can take days or weeks because of large data set.
- **Recognizing Different Texts**: The model has to learn to read many different fonts, languages, and even handwriting, which makes it harder
- **Fixing Mistakes**: Sometimes the model makes mistakes, and you have to spend time correcting them.
- **Time**: All of this takes a lot of time and effort's making it a tough task.

Why using Tesseract OCR is a better choice :-

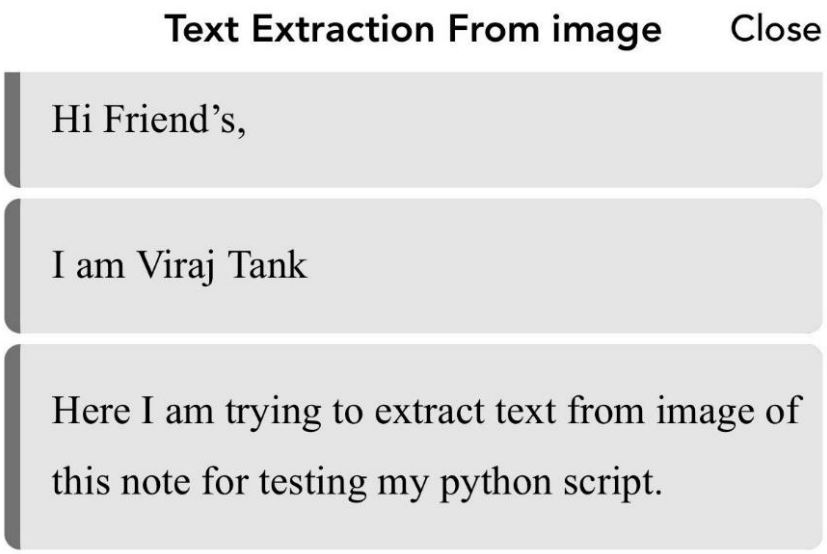
- **No Need for Data Collection**: Tesseract already knows how to read text, so you don't need to gather and label thousands of images.
- **Fast Setup**: You can start using Tesseract right away, without waiting for days or weeks to train a model.
- **Works with Different Texts**: Tesseract can read many different fonts, languages, and even handwriting without extra effort.
- **Less Mistakes**: Tesseract has been tested a lot, so it makes fewer mistakes and works well out of the box.
- **Saves Time**: Using Tesseract is free, quick to set up, and doesn't require much effort, saving time.

Implementation Of Tesseract OCR :-

```
import cv2
import pytesseract

# Function to extract text from an image
def extract_text_from_image(image_path):
    # Read the image using OpenCV
    img = cv2.imread(image_path)
```

```
    # Use pytesseract to extract text from the pre-processed image
    text = pytesseract.image_to_string(img)

    return text

# Image path
image_path = '/content/drive/MyDrive/Task3/test_image/Final Printed Text
Image.jpg'

# Extract text
extracted_text = extract_text_from_image(image_path)

# Display the extracted text
print("Extracted Text:")
print(extracted_text)
```

Input Image :-



Results :-

```
Extracted Text:
Text Extraction From image Close

| Hi Friend's,
I am Viraj Tank

Here I am trying to extract text from image of

this note for testing my python script.
```

Observation :-

In Extracted Text we can see that we are getting '|' before 'Here' which is not initially present in the Input Image.

Conclusion :-

Making my own system to read text from pictures is very hard and takes a lot of time and work. I need to gather lots of pictures, teach the computer to read them, and fix any mistakes it makes. Instead of doing all this hard work, I can use a tool called Tesseract OCR, which is already good at reading text from pictures. Tesseract is easy to use, free, and can read many different types of text. I tried using Tesseract in a simple Python script to get text from pictures, but I had some trouble getting the right result. Even with this small problem, Tesseract is still a great tool to use because it saves a lot of time and effort.

Task 4 :- Text Extraction From An Image And Improving The Accuracy

Introduction :-

In this task, I used Python to extract text from images in two ways: first without preprocessing and then with preprocessing steps like grayscale conversion, blurring, and thresholding. By comparing the results, I observed how preprocessing helps the computer read text more accurately. Preprocessing is the process of cleaning and improving the image before passing it to the OCR tool. It helps make the text in the image clearer and easier for the computer to understand. Common preprocessing steps include converting the image to grayscale, applying blurring to reduce noise, and using thresholding to make the text stand out more.

Method and WorkFlow :-

- Input Image :-
  An image containing text is selected as the input.
- Preprocessing the Image :-
  To improve the image quality for OCR, the following preprocessing steps are applied:
  - Grayscale Conversion :- The image is converted to grayscale, removing colors. This helps reduce unnecessary information that may confuse.
  - Gaussian Blur :- A blurring filter is applied to smooth the image and reduce noise. Noise such as random pixels or distortions is minimized, making the text clearer.
  - Thresholding :- This makes the text stand out more distinctly against the background.
  - Character Filtering :- After text extraction, the output is cleaned by removing unwanted symbols and keeping only valid characters like letters and numbers.
- Text Extraction with Preprocessing :-
  The preprocessed image is passed to Tesseract OCR.
- Text Extraction Without Preprocessing :-
  The original image is directly passed to Tesseract OCR without applying any modifications or enhancements.
- Accuracy Comparison :-

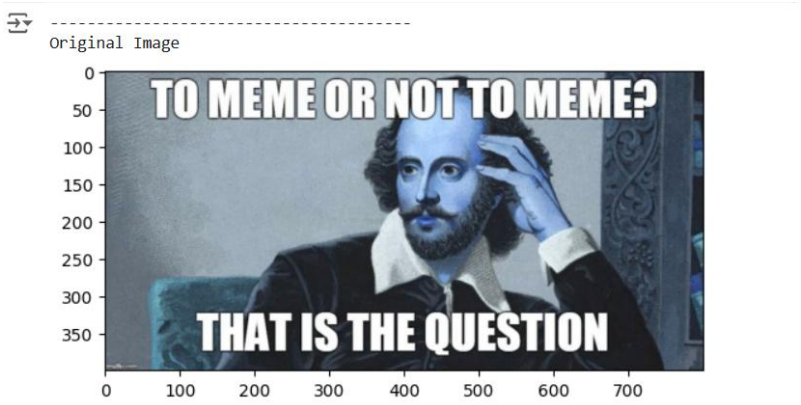
The accuracy of the text extracted is measured by comparing it with the actual text using the SequenceMatcher library.

Combining Preprocessing & Tesseract :-

```
!pip install opencv-python pytesseract -q
!apt-get install tesseract-ocr -q

import json
import cv2
import numpy as np
import pytesseract
from matplotlib import pyplot as plt
import string
from difflib import SequenceMatcher

def preprocessing(image):
    print("--------------------------------------")
    print("Original Image")
    plt.imshow(image)
    plt.show()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = ((gray / 255) ** 20 * 255).astype(np.uint8)
    print("--------------------------------------")
    print("Gray Scale Image")
    plt.imshow(gray)
    plt.show()
    blur = cv2.GaussianBlur(gray, (3, 3), 0)
    thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)[1]
    print("--------------------------------------")
    print("Thresholded Image")
    plt.imshow(thresh)
    plt.show()
    data = pytesseract.image_to_string(thresh, lang='eng', config='--psm
6')
    allowed = string.ascii_uppercase + string.ascii_lowercase + " ?"
    lines = data.split("\n")
    res = []
    for line in lines:
        t = []
        correct = 0
        for i in line:
            if i in allowed:
                t.append(i)
                correct += 1
            elif i in "1234567890":
                t.append(i)
            elif len(t) > 0 and t[-1] != " ":
                t.append(" ")
        if correct > len(line) * 3 / 5 and correct > 2:
            res.append("".join(t))
```

```
    data = "\n".join(res)
    return data

def process(image):
    a = preprocessing(image)
    return a

# Path to image file
image_path = "/content/drive/MyDrive/Task3/test_image/test-7.jpg"
#image_path = "/content/drive/MyDrive/Task3/test_image/ddu_3.png"  #
Update this with your image's path

# Actual text for comparison
actual_text = "TO MEME OR NOT TO MEME?\n THAT IS THE QUESTION"

# Read and process the image
img = cv2.imread(image_path)

processed_output = process(img)

# Tesseract prediction
tesseract_output = pytesseract.image_to_string(image_path, lang='eng',
config='--psm 6')
print("----------------------------------------")
print("Tesseract Prediction \n" + tesseract_output)

# Custom processing and prediction
print("----------------------------------------")
print("Processed Output Text \n" + processed_output)

# Compare the results
tesseract_ratio = SequenceMatcher(None, actual_text,
tesseract_output).ratio()
processed_ratio = SequenceMatcher(None, actual_text,
processed_output).ratio()

print("----------------------------------------")
print("Tesseract Accuracy Ratio = {:.2f}".format(tesseract_ratio))
print("Custom Processing Accuracy Ratio = {:.2f}".format(processed_ratio))
```

Results :-



```
---------------------------------------
Gray Scale Image
```


```
---------------------------------------
Thresholded Image
```


```
----------------------------------------
Tesseract Prediction
__TO\MEMEOR|NOT,TO\MEME?
- rmN |
WMS si

----------------------------------------
Processed Output Text
TO MEME OR NOT TO MEME?
THAT IS THE QUESTION


----------------------------------------
Tesseract Accuracy Ratio = 0.53
Custom Processing Accuracy Ratio = 0.99
```

Observation :-

- The accuracy ratio of text extracted directly using Tesseract OCR was only 0.53.
- After applying preprocessing techniques like grayscale conversion, Gaussian blur, and thresholding, the accuracy ratio improved to 0.99.

Conclusion :-

In conclusion, this test shows that cleaning up the image before using OCR makes a big difference. Without any changes to the image, the OCR got only 53% of the text right. But when we made the image clearer by changing it to black and white, smoothing it, and making the text sharper, the OCR got 99% of the text right. This proves that preparing the image before using OCR helps the computer read the text much more accurately. Because of this, Tesseract is mainly used to extract text from scanned or printed copies, where the text is clear, and not from coloured images where the text might be harder to read.