NAME: BHAVYA TANEJA

MILESTONE-2 REPORT

BFSI- OCR OF BANK STATEMENTS

Objective:

Implement and Fine-tune OCR Capabilities to Extract Data Accurately from Bank Statements.

Introduction:

In this milestone, I focused on implementing and fine-tuning Optical Character Recognition (OCR) capabilities to extract data accurately from bank statements. Given the critical nature of precision in extracting financial information, I explored various OCR techniques, including EasyOCR, PaddleOCR, and Tesseract. The objective was to assess their performance and determine the most suitable solution for accurate text extraction from scanned bank statements. Additionally, I created a user interface (UI) for an OCR comparator using Gradio to facilitate easy text comparison for OCR results.

OCR Techniques Evaluation:

1. EasyOCR: EasyOCR is an open-source OCR library developed by the jaidedai team. It supports over 80 languages and is known for its speed and high accuracy, particularly in recognizing text in documents with mixed fonts and noisy backgrounds. It utilizes deep learning-based models for character recognition and has been widely praised for its simplicity in implementation.

- **Strengths:**
    - Supports a wide variety of languages.
    - Can handle complex documents and noisy backgrounds.
    - Faster than many alternatives due to optimized model architecture.
- **Weaknesses:**
    - Might require fine-tuning on highly specialized documents for maximum accuracy.

2. PaddleOCR: PaddleOCR is a part of the PaddlePaddle deep learning framework. It is designed to handle both text detection and recognition in images and supports multiple languages. PaddleOCR boasts an extensive suite of pre-trained models for detecting various fonts and layouts of text.

- **Strengths:**
    - Robust in recognizing a wide range of text in different languages and fonts.
    - Pre-trained models are effective for standard OCR use cases.
- **Weaknesses:**
    - Performance can degrade with highly complex or low-quality images, requiring additional optimization and tuning.
    - Less efficient on resource-constrained systems compared to alternatives like EasyOCR.

3. Tesseract: Tesseract is one of the most well-known and widely used OCR engines. It is an open-source OCR tool developed by Google and has been in development for many years. Tesseract supports over 100 languages and provides a comprehensive set of features for OCR tasks, including page layout analysis.

- **Strengths:**
    - Mature and highly customizable, especially for specialized use cases.
    - Strong community support and extensive documentation.
- **Weaknesses:**
    - Performance can be slower compared to newer OCR models.
    - Tuning and pre-processing may be required for better accuracy with noisy or poor-quality images.

Performance Analysis:

After testing these OCR techniques on a variety of bank statement images with varying text qualities and layouts, I concluded that **EasyOCR** and **Tesseract** provided better results than PaddleOCR. Specifically:

- **EasyOCR** showed the best overall performance in terms of speed and accuracy, successfully extracting information from most bank statements with minimal errors. It also handled different fonts and layouts effectively.

- **Tesseract**, although slower than EasyOCR, showed robust results in extracting data from complex bank statements with mixed fonts and images.

UI Design with Gradio:

After selecting the most suitable OCR techniques, I focused on designing a user interface (UI) that would allow users to upload images, extract text, and compare results from different OCR models. For this, I used **Gradio**, a Python library for creating customizable UIs for machine learning models.

- **Gradio Features Used:**
    - Easy-to-create interface for uploading images and displaying results.
    - Support for multiple inputs and outputs, which allowed me to create an OCR comparator.
    - Interactive UI that displays the extracted text alongside comparison results from multiple OCR models.

UI Workflow:

1. **User Uploads Image:**
    - The user uploads a scanned bank statement or any other image containing text.
2. **OCR Processing:**
    - The image is passed through the selected OCR models (EasyOCR, Tesseract).
3. **Comparison Display:**
    - The extracted text from each OCR technique is displayed side by side for comparison.
    - Users can easily assess which OCR model gave the most accurate results for their particular image.

CODE USED:

```
import cv2
import numpy as np
import easyocr
import pytesseract
import gradio as gr
import csv
import os

# Example function to detect and recognize text using EasyOCR and Pytesseract
def detect_and_recognize(image, min_size, text_threshold, low_txt, link_threshold):
    # Initialize OCR models
    easyocr_reader = easyocr.Reader(['en'])

    # EasyOCR text detection and recognition
    easyocr_output = easyocr_reader.readtext(image)
    easyocr_boxes = [box[0] for box in easyocr_output]
    easyocr_text = [box[1] for box in easyocr_output]

    # Pytesseract text detection and recognition
    pytesseract_output = pytesseract.image_to_data(image,
output_type=pytesseract.Output.DICT)
    pytesseract_boxes = []
    pytesseract_text = []

    for i in range(len(pytesseract_output['text'])):
        if int(pytesseract_output['conf'][i]) > 0:  # Only consider text with positive confidence
            pytesseract_boxes.append((
```

```
            pytesseract_output['left'][i],
            pytesseract_output['top'][i],
            pytesseract_output['left'][i] + pytesseract_output['width'][i],
            pytesseract_output['top'][i] + pytesseract_output['height'][i]
        ))
        pytesseract_text.append(pytesseract_output['text'][i])

    return {
        'EasyOCR': {'boxes': easyocr_boxes, 'text': easyocr_text},
        'Pytesseract': {'boxes': pytesseract_boxes, 'text': pytesseract_text}
    }

# Function to draw bounding boxes on the image
def draw_boxes(image, easyocr_boxes, pytesseract_boxes):
    easyocr_image = image.copy()
    pytesseract_image = image.copy()

    for box in easyocr_boxes:
        pts = np.array(box, np.int32)
        pts = pts.reshape((-1, 1, 2))
        easyocr_image = cv2.polylines(easyocr_image, [pts], isClosed=True, color=(0, 255, 0), thickness=2)

    for box in pytesseract_boxes:
        pts = np.array([[box[0], box[1]], [box[2], box[1]], [box[2], box[3]], [box[0], box[3]]], np.int32)
        pts = pts.reshape((-1, 1, 2))
```

```
        pytesseract_image = cv2.polylines(pytesseract_image, [pts], isClosed=True,
color=(255, 0, 0), thickness=2)


    return easyocr_image, pytesseract_image


# Function to compare OCR results with EasyOCR and Pytesseract
def compare_ocr(image, min_size, text_threshold, low_txt, link_threshold, canvas_size):
    # Ensure that canvas_size is a tuple of two positive integers
    if isinstance(canvas_size, tuple) and len(canvas_size) == 2:
        width, height = canvas_size
        if width > 0 and height > 0:
            image = cv2.resize(image, (width, height))  # Resize the image if canvas_size is valid
    else:
        # Handle the case where canvas_size is invalid
        print("Invalid canvas size. No resizing applied.")

    # Convert the image to a format that OpenCV can use (numpy array)
    image = np.array(image)


    # Get OCR results
    output = detect_and_recognize(image, min_size, text_threshold, low_txt, link_threshold)


    # Draw bounding boxes for both OCR methods
    easyocr_image, pytesseract_image = draw_boxes(image, output['EasyOCR']['boxes'],
output['Pytesseract']['boxes'])


    # Prepare text results for both OCR methods
    text_results_easyocr = [[str(i + 1), text] for i, text in enumerate(output['EasyOCR']['text'])]
```

```
    text_results_pytesseract = [[str(i + 1), text] for i, text in
enumerate(output['Pytesseract']['text'])]


    # Save text results to CSV
    csv_filename = "ocr_results.csv"
    with open(csv_filename, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Index", "EasyOCR Text", "Pytesseract Text"])
        for easyocr_text, pytesseract_text in zip(text_results_easyocr,
text_results_pytesseract):
            writer.writerow([easyocr_text[0], easyocr_text[1], pytesseract_text[1]])


    # Return images with bounding boxes, text results, and the CSV file
    return easyocr_image, pytesseract_image, text_results_easyocr,
text_results_pytesseract, csv_filename


# Gradio interface for user interaction
def create_interface():
    # Parameters for OCR
    min_size = gr.Slider(minimum=1, maximum=10, value=5, label="Minimum Size")
    text_threshold = gr.Slider(minimum=0, maximum=1, step=0.01, value=0.5, label="Text
Threshold")
    low_txt = gr.Slider(minimum=0, maximum=1, step=0.01, value=0.4, label="Low Text
Threshold")
    link_threshold = gr.Slider(minimum=0, maximum=1, step=0.01, value=0.4, label="Link
Threshold")
    canvas_size = gr.Slider(minimum=1, maximum=1000, step=1, value=500, label="Canvas
Size", interactive=True)
```

```
    # Gradio interface with a submit button
    iface = gr.Interface(
        fn=compare_ocr,
        inputs=[
            gr.Image(type="numpy", label="Upload Image"),
            min_size,
            text_threshold,
            low_txt,
            link_threshold,
            canvas_size
        ],
        outputs=[
            gr.Image(type="numpy", label="EasyOCR Bounded Image"),
            gr.Image(type="numpy", label="Pytesseract Bounded Image"),
            gr.Dataframe(label="EasyOCR Text Results"),
            gr.Dataframe(label="Pytesseract Text Results"),
            gr.File(label="Download CSV")
        ],
        live=False  # Disable live processing, use submit button instead
    )
    return iface

# Run the interface
iface = create_interface()
iface.launch(share=True)
```

Technical Information: Libraries Used in the OCR Comparison Project

1. **OpenCV (cv2)**:
    - **Purpose**: OpenCV (Open Source Computer Vision Library) is used for image processing tasks. In this project, it helps in handling image loading, resizing, and drawing bounding boxes around detected text.
    - **Key Functions**:
        - cv2.polylines: Used to draw bounding boxes around the detected text areas.
        - cv2.resize: Resizes the input image to match the canvas size for better display and processing.
2. **NumPy**:
    - **Purpose**: NumPy is a powerful library for numerical computations. It is used in this project to handle images as arrays, which is essential for OpenCV processing.
    - **Key Functions**:
        - np.array: Converts the input image to a NumPy array, which is compatible with OpenCV.
        - np.int32: Defines the type for the bounding box points.
3. **EasyOCR**:
    - **Purpose**: EasyOCR is an optical character recognition (OCR) library that is used to detect and recognize text in images. It is particularly effective for detecting text in various fonts and noisy images.
    - **Key Functions**:
        - easyocr.Reader: Initializes the OCR model, where the language parameter (['en'] for English) is specified.
        - reader.readtext: Performs both text detection and recognition from the image.
4. **Pytesseract**:
    - **Purpose**: Pytesseract is a Python wrapper for Google's Tesseract-OCR engine. It is used in this project to detect and extract text from images.

- **Key Functions**:
    - pytesseract.image_to_data: Extracts detailed information from the image, such as the detected text, confidence scores, and bounding box coordinates.

5. **Gradio**:
    - **Purpose**: Gradio is a library for creating interactive UIs for machine learning models. In this project, it is used to build a user interface that allows users to upload images, view OCR results, and compare outputs from EasyOCR and Pytesseract.
    - **Key Functions**:
        - gr.Interface: Creates a user interface to interact with the OCR comparison function, allowing image uploads and displaying results.
        - gr.Slider: Used to create interactive sliders for various OCR parameters (e.g., minimum size, text threshold).
        - gr.Image, gr.Dataframe, and gr.File: Used to display images, show the OCR results in tabular format, and allow the user to download a CSV file with the OCR results.
6. **CSV**:
    - **Purpose**: The csv module is used to write the OCR results into a CSV file for further analysis and comparison.
    - **Key Functions**:
        - csv.writer: Writes the OCR results (EasyOCR and Pytesseract text) to a CSV file, including an index for each text element.
7. **os**:
    - **Purpose**: The os module is used to handle file system operations such as checking and creating files or directories.
    - **Key Functions**:
        - os.path.exists: Can be used for file existence checks, although it's not explicitly used in the provided code, it could be utilized for additional file handling.

CONCLUSION :

This milestone was highly beneficial as it provided valuable insights into various OCR techniques, including EasyOCR, Pytesseract, and PaddleOCR, and how to implement them for text recognition. Through hands-on experience, I learned the strengths and limitations of each technique in extracting accurate data from images. Additionally, I gained practical knowledge in creating a user-friendly interface using Gradio, enabling efficient comparison of OCR results. Overall, this milestone enhanced my understanding of OCR models and UI development, equipping me with the skills to integrate these techniques into real-world applications effectively.