DAY 1 (Date:09/06/2025)
TASK 1:

What is OCR?

Optical Character Recognition (OCR) is a technology that allows computers to **recognize and convert text from images or scanned documents into machine-readable text**.

This means if you scan a printed or handwritten answer sheet, OCR can extract the actual words and numbers into editable digital text.

How OCR Works

OCR involves several key steps:

1. **Image Acquisition**
   The input is typically a scanned document or image captured by a camera.
2. **Preprocessing**
   The image is cleaned to improve recognition:
   - **Grayscale conversion**
   - **Noise removal**
   - **Thresholding (binarization)**
   - **Resizing or smoothing**
   - **Deskewing (correcting tilt)**
3. **Text Detection**
   The image is scanned to locate blocks or lines of text.
4. **Character Segmentation**
   The system splits lines into individual words and characters.
5. **Feature Extraction**
   Each character is analyzed for distinctive patterns and shapes.
6. **Character Recognition**
   Recognized using:
   - **Matrix matching (template-based)**

    - **Machine learning or deep learning models (like LSTM)**
7. **Post-processing**
    - Spelling correction
    - Formatting restoration
    - Mapping characters to a language dictionary

Common Challenges in Document Extraction

1. **Low Image Quality**
    - Blurry, skewed, or low-resolution images can reduce accuracy.
2. **Handwriting Recognition**
    - Handwritten characters vary widely in shape and style.
    - Harder to interpret than printed text.
3. **Complex Layouts**
    - Tables, diagrams, and multiple columns make it hard to detect reading order.
    - Answer sheets often include mixed content (text, formulas, boxes).
4. **Noise and Artifacts**
    - Marks, stains, shadows, or fold lines interfere with character detection.
5. **Language and Fonts**
    - Multilingual content and stylized fonts can confuse OCR models.
6. **Mathematical Symbols**
    - Symbols like ∑, √, ≤ are not easily recognized by standard OCR.

Technologies Used in OCR

- **Tesseract OCR**: An open-source OCR engine developed by Google. Supports over 100 languages and works well with printed text.

- **Leptonica**: Image processing library often used with Tesseract.
- **Deep Learning Models**: LSTM, CNN, and transformer-based models are used in modern OCR pipelines (like Google Vision, AWS Textract, etc.).

What I Have Learned

- The **core steps and logic behind OCR**.
- The **importance of preprocessing** for improving accuracy.
- How **different types of content (printed vs handwritten)** affect results.
- Implemented a **mini OCR project using Python + Tesseract**.
- Learned how to extract and save text from a scanned image efficiently.
- Understood how OCR can be used in **AI-driven education platforms** to automate and scale assessment processing.

TASK 2:

1. **Theory-Based Answer Sheets**
    - These contain continuous text, written explanations, and structured paragraphs.
    - Layout considerations include margins, spacing, and paragraph structure.
    - Challenges in OCR processing: Handwritten text may require advanced recognition models to maintain accuracy.
2. **Mathematical Answer Sheets (With Equations)**
    - These include complex formulas, calculations, and handwritten equations.
    - Formatting varies: Some have neat alignment, while others mix text with equations.
    - OCR challenges: Recognizing mathematical symbols, fractions, and formulas accurately without distortion.
3. **Diagram-Based Answer Sheets**

- These feature sketches, graphs, labeled illustrations, and flowcharts.
- Mixed content structure: Diagrams may be accompanied by explanatory text or annotations.
- OCR considerations: Diagrams are difficult to interpret directly—advanced image processing or manual verification may be needed.

What I Learned

- OCR performance varies **greatly depending on the type of answer sheet**.
- Preprocessing must be **customized for each layout type** to improve accuracy.
- Theory-based sheets work best with standard OCR and preprocessing.
- Mathematical sheets require **symbol-aware OCR** or LaTeX extraction tools.
- Diagrams need **hybrid approaches** combining OCR and image analysis.
- Understanding these structures helps build **more intelligent document extraction systems**, especially for academic automation.

Mini-Project: OCR-Based Image processing

Objective:

The goal of this project was to explore the application of **Optical Character Recognition (OCR)** techniques to efficiently extract text from scanned documents and images.

Methodology:
OCR-based text extraction code:

1. Importing Required Libraries

```
import cv2

from PIL import Image

import pytesseract
```

- cv2: OpenCV is used for image processing.
- PIL: The Python Imaging Library (Pillow) is used for handling images.

- pytesseract: The Tesseract OCR library for text extraction.

2. Setting Up Tesseract OCR

pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

- Specifies the **path to Tesseract OCR** on the system.
- Ensures Python can access and run OCR functionality.

3. Preprocessing the Image for OCR

```
def preprocess_image(img_path):
    print(f"[INFO] Loading image: {img_path}")
    img = cv2.imread(img_path)


    if img is None:
        print("[ERROR] Image not found!")
        return None
```

- Loads the image using OpenCV (cv2.imread()).
- Handles errors if the image is missing.

Image Processing Steps

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- Converts the image to **grayscale**, which improves OCR accuracy.

```
    gray = cv2.resize(gray, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
```

- **Resizes** the image to make text clearer.

```
    blurred = cv2.bilateralFilter(gray, 9, 75, 75)
```

- **Reduces noise** while preserving edges.

```
    _, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

- Applies **Otsu's thresholding**, which separates text from the background.

4. Extracting Text Using Tesseract OCR

```
def extract_text(preprocessed_img):
    if preprocessed_img is None:
        return "[ERROR] No image to extract text from."
```

- Checks if the processed image exists before proceeding.

```
    pil_img = Image.fromarray(preprocessed_img)
```

- Converts the OpenCV image into a **PIL format**, which Tesseract can read.

```
    custom_config = r'--oem 3 --psm 6'
    text = pytesseract.image_to_string(pil_img, config=custom_config)
```

- Uses **OCR Engine Mode (--oem 3)** for better recognition.
- Uses **Page Segmentation Mode (--psm 6)** for mixed text types.

5. Running the Program

```
if __name__ == "__main__":
    image_path = r'C:\Users\nsaba\OneDrive\Pictures\workshop.jpg'

    processed_img = preprocess_image(image_path)
    extracted_text = extract_text(processed_img)

    print("\n----- Extracted Text -----\n")
    print(extracted_text)
```

- **Loads the image**, processes it, and extracts text.
- Prints the extracted text.

6. Saving the Extracted Text

```
    with open("output.txt", "w", encoding="utf-8") as f:
        f.write(extracted_text)
```

- Saves the extracted text into a **text file (output.txt)**.

Summary of What This Code Does

- Loads an image.
- Enhances it for better OCR accuracy.
- Uses Tesseract OCR to extract text.
- Saves extracted text into a file.

PROGRAM:

```
import cv2
from PIL import Image
import pytesseract
# Set Tesseract command path
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
def preprocess_image(img_path):
    print(f"[INFO] Loading image: {img_path}")
    img = cv2.imread(img_path)
    if img is None:
        print("[ERROR] Image not found!")
        return None
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Resize to improve accuracy (scale text)
    gray = cv2.resize(gray, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
    # Smooth image using bilateral filter
    blurred = cv2.bilateralFilter(gray, 9, 75, 75)
    # Apply Otsu's thresholding
    _, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return thresh
```

```
def extract_text(preprocessed_img):
    if preprocessed_img is None:
        return "[ERROR] No image to extract text from."
    pil_img = Image.fromarray(preprocessed_img)
    # Custom OCR config to improve results
    custom_config = r'--oem 3 --psm 6'
    text = pytesseract.image_to_string(pil_img, config=custom_config)
    return text
if __name__ == "__main__":
    # Update the image file name here
    image_path = r'C:\Users\nsaba\OneDrive\Pictures\workshop.jpg'
    processed_img = preprocess_image(image_path)
    extracted_text = extract_text(processed_img)
    print("\n----- Extracted Text -----\n")
    print(extracted_text)
    # Save result to file
    with open("output.txt", "w", encoding="utf-8") as f:
        f.write(extracted_text)
```

INPUT:

OUTPUT:

GENERATIVE Al“ s

4)

GENERATIVE Al

"° DATE MENTOR

Sunday MS.VAISHNAVI N

VS)

Thank you