Task 1

I can use deep learning to train a model that predicts the rotation angle for an OCR page. However, this process can be time-consuming and requires significant effort due to the need for data annotation and model training. Therefore, I decided to develop the code based on familiar techniques commonly used to identify or correct skew or rotation in scanned and image documents. Techniques such as the Hough Line Transform, Tesseract OCR, and adaptive thresholding are part of the state-of-the-art methods available in computer vision and document preprocessing, using a combination of approaches:

1. **Hough Line Transform:** is used to detect straight lines in the document and figure out rotation angles based on the skew of the detected lines.**How detect Rotation Angle Using Hough Line Transform:**
    - **Convert the Image to Grayscale**:
        - For line detection, we first convert the image to grayscale to simplify the processing.
    - **Edge Detection**:
        - Use edge detection (e.g., **Canny Edge Detection**) to highlight the edges in the image. This helps in detecting the lines more effectively.
    - **Apply Hough Line Transform**:
        - Apply the Hough Line Transform to detect lines in the image. The transform will return the parameters of lines in polar coordinates (ρ and θ).
        - The angle θ will represent the orientation of each detected line.
    - **Calculate the Median Angle**:
        - Extract the angles θ of the detected lines.
        - Normalize these angles to a common range (e.g., [-45, 45] degrees).

    - Compute the **median** of the detected angles, which represents the dominant angle of skew in the image.
- **Correct the Image Rotation**:
    - Based on the calculated skew angle, you can rotate the image back to an upright orientation.

2. **Tesseract OSD (Orientation and Script Detection):** Detect text orientation when line Hough Line Transform fails.

3. **Manual Rotation tries common angles (90, 180, 270 degrees)** if both previous methods fail.

The code was written in Python, leveraging libraries like **OpenCV** for image processing, **pytesseract** for OCR, and **PyPDF2** for handling PDF documents.

The code should work on a variety of PDF files, especially those containing scanned or image-based documents that may need rotation correction.

Task 2

The code was developed based on the process of classifying PDF pages into three categories by analyzing text extraction from the pages using a combination of methods:

1. **Text Extraction via PyPDF2**: The first step involves trying to extract text directly from the PDF using the PyPDF2 library. This works for **machine-readable.**
2. **OCR (Optical Character Recognition)**: If the extracted text is too short (less than 31 characters, typically metadata), the page is assumed to be scanned or image-based, and OCR is applied using Tesseract to read the text from the image.

3. **Classification**:
    - **0 (Machine-readable)**: If the extracted text by PyPDF2 is long enough to indicate that the page contains text (length > 31 characters).
    - **1 (OCR-able)**: If OCR successfully extracts text from image-based or scanned pages.
    - **2 (Not OCR-able)**: If both methods fail to extract any text.

Development Basis:

- **PyPDF2** is used to work with PDFs and extract text (**Machine-readable**).
- **Pytesseract** (which is an interface for the Tesseract OCR engine) is used to perform OCR on image-based pages.
- **OpenCV** is used for image preprocessing to enhance the quality of the OCR process, by converting the image to grayscale and applying adaptive thresholding to make the text clearer for the OCR engine.
- **Pdf2image** is used to convert PDF pages to images, making it possible to perform image-based processing on them.

It is a generalized solution and is suitable for use with different PDF documents but specifically to the documents that have both the machine extractable text and scanned images. It has been designed to sort and process each page as per its requirements.

Cherry on the Cake Task

For PDF Document Partitioning, the code was developed to handle a **200-page PDF** containing both **machine-readable** and **scanned documents**. The process works as follows:

Key Components:

1. **Text Extraction with PyPDF2**:
    - First, the code attempts to extract text from each page. If successful, it uses patterns like "Document X" to group pages.

2. **PDF to Image Conversion**:
    - For non-machine-readable pages, the code converts them into images for **visual analysis** using **pdf2image**.
3. **OCR Fallback:**:
    - If text extraction detection fail, **OCR Tesseract** is used to extract text from scanned pages.
4. **Visual Feature Detection with OpenCV**:
    - If both text extraction and OCR detection fail, the code uses visual cues like **colored borders**, **headers/footers**, and **watermarks** to detect document boundaries.

Generalization:

- The solution works with any PDF, using both **text content** and **visual layout** to detect document transitions.

References

1. **Duda, R.O., Hart, P.E.**
   "Use of the Hough Transformation to Detect Lines and Curves in Pictures,"
   Communications of the ACM, Vol. 15, No. 1, 1972, pp. 11-15.
   DOI: 10.1145/361237.361242