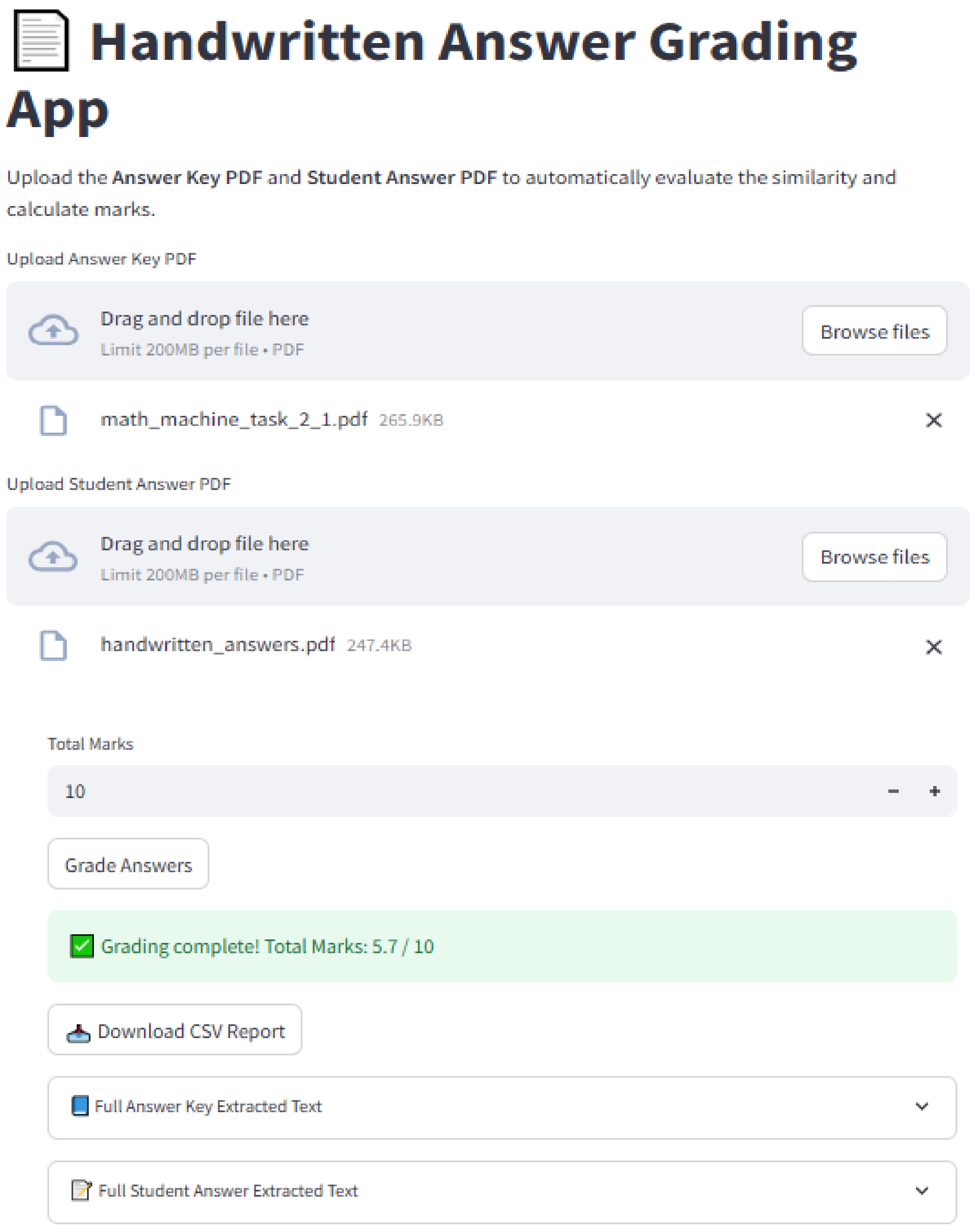
Handwritten-Answer-Grading-App

Machine_task_2

Github link : https://github.com/abhiramkrishnayanam/Handwritten-Answer-Grading-App



Handwritten Answer Grading App

Upload the **Answer Key PDF** and **Student Answer PDF** to automatically evaluate the similarity and calculate marks.

Upload Answer Key PDF

Drag and drop file here
Limit 200MB per file • PDF

math_machine_task_2_1.pdf 265.9KB

Upload Student Answer PDF

Drag and drop file here
Limit 200MB per file • PDF

handwritten_answers.pdf 247.4KB

Total Marks

10

✅ Grading complete! Total Marks: 5.7 / 10

Download CSV Report
Full Answer Key Extracted Text
Full Student Answer Extracted Text

Overview

This project provides an automated system to grade handwritten answers by comparing scanned student answer PDFs against an answer key PDF. The system extracts text from images of PDF pages using OCR, computes similarity scores between student answers and the answer key, and assigns marks accordingly. The entire process is accessible through a simple web interface built with Streamlit.

Tools and Libraries Used

- **PyMuPDF (fitz):** To render PDF pages as high-resolution images.
- **OpenCV:** For image preprocessing to enhance OCR accuracy.
- **PIL (Pillow):** For image manipulation.
- **Tesseract OCR:** To extract text from images.
- **FuzzyWuzzy:** To calculate similarity scores between answer texts.
- **Streamlit:** For building the user-friendly web application interface.

What I have done here :

To simplify testing, I limited the Answer Key document to only the first page**,** as the sample handwritten answers were written on that page alone.
Both the Answer Key PDF and the Handwritten Answer PDF were then converted into images using **PyMuPDF.**
I applied OCR (Optical Character Recognition) via **pytesseract** to extract text from these images.
After that, the extracted texts were cleaned by removing spaces, newlines, and punctuation for accurate comparison.
Finally, I used the **fuzz.partial_ratio()** method from the **fuzzywuzzy** library to compute the similarity percentage between the processed student answer and the corresponding answer key, enabling a score-based evaluation.

Step-by-Step Approach

1. **PDF Rendering to Images**
   The system converts each page of both the answer key and student answer PDFs into high-resolution images. This allows working with image data since the answers are handwritten.
2. **Image Preprocessing and OCR**
   Each rendered page image undergoes preprocessing steps such as grayscale conversion, thresholding, and denoising to improve OCR text extraction quality. Tesseract OCR is then applied to extract textual content from these images.

3. **Text Preprocessing**
   Extracted texts are cleaned by converting to lowercase, removing whitespace, newlines, and punctuation to enable a fair comparison regardless of formatting variations.

4. **Similarity Scoring and Grading**
   The cleaned student answer texts are compared to the corresponding answer key texts using fuzzy string matching (partial ratio). Marks are assigned proportionally based on similarity scores for each question.

5. **Output Generation**
   The system generates a detailed CSV report containing each question's extracted texts, similarity percentages, and marks awarded. It also saves the full extracted texts for reference.

6. **User Interface**
   A Streamlit-based web app allows users to upload the answer key and student answer PDFs, specify total marks, run the grading pipeline, view results, and download the grading report.

Execution Workflow with Module and Method References

1. **PDF Upload and Temporary Storage**
   Module: `app.py`
   Methods Used:
   - `st.file_uploader()` – Uploads the Answer Key and Student PDFs.
   - `tempfile.TemporaryDirectory()` – Creates a temporary folder for storing files during the session.
   - `open().write()` – Saves the uploaded files locally for further processing.

2. **PDF to Image Conversion**
   Module: `pdf_to_images.py`
   Function Called:
   - `render_pdf_pages_to_images(pdf_path, output_folder)`
     Converts each page of the PDFs into high-resolution images using the `fitz` library (PyMuPDF).

3. **Image Preprocessing and OCR (Text Extraction)**
   Module: `ocr_utils.py`
   Functions Called:
   - `extract_text_from_images(image_folder)`
     Iterates over each image file.

        - Internally calls `preprocess_image(image_path)` to convert to grayscale, apply thresholding, and denoise the image using OpenCV.
        - Uses `pytesseract.image_to_string()` to extract text from the processed image.

4. **Text Cleaning and Similarity Calculation**
   Module: `grading_pipeline.py`
   Functions Called:
    - `preprocess_text(text)`
      Normalizes the text by removing whitespace, punctuation, and converting it to lowercase.
    - `fuzz.partial_ratio()` from the `fuzzywuzzy` library
      Calculates the similarity percentage between cleaned answer key text and student text.

5. **Marks Calculation and Report Generation**
   Module: `grading_pipeline.py`
   Functions Called:
    - `grade_answers(answer_key_folder, student_folder, total_marks)`
      Computes similarity scores, calculates marks per question, and totals the score.
        - Generates a `grading_report.csv` file for all results.
    - `save_texts_to_file(texts, filename)`
      Saves the raw OCR outputs into text files for the answer key and student responses.

6. **Streamlit UI: Display and Download Reports**
   Module: `app.py`
   Methods Used:
    - `st.success()` – Displays the total marks obtained.
    - `st.download_button()` – Allows downloading the CSV report.
    - `st.expander()` – Lets users view the full extracted text for both answer key and student responses.

Challenges and Resolutions:

Extracting Text from Handwritten Answers

Handwritten content is often noisy and irregular, making OCR difficult. To address this, preprocessing techniques such as thresholding and denoising are applied before OCR.

Handling Variations in Text Formatting

Differences in spacing, capitalization, and punctuation can affect similarity calculations. The system cleans texts by removing these variations for consistent comparison.

Matching Corresponding Pages Between PDFs

The system assumes answer key and student PDFs have the same number of pages/questions and processes them sequentially.

Resource Management

Temporary files and directories are handled carefully to avoid clutter during batch processing.

Possible Improvements

- Incorporate more advanced image preprocessing techniques to further enhance OCR accuracy on challenging handwritten text.
- Integrate deep learning-based handwriting recognition models to improve text extraction quality.
- Add a manual correction interface for users to edit OCR-extracted texts before grading.
- Support question-wise marking schemes and partial credit for multi-part answers.
- Extend compatibility for multi-column PDFs or answer sheets with complex layouts.

Setup Instructions

1. **Install Dependencies**
   Install the required Python packages, including `PyMuPDF`, `opencv-python`, `pillow`, `pytesseract`, `fuzzywuzzy`, and `streamlit`.

2. **Install Tesseract OCR Engine**
   Download and install Tesseract OCR from its official repository. Update the script with the correct executable path to enable OCR functionality.

3. **Run the Streamlit Application**
   Launch the Streamlit app with the command:

   ```
   streamlit run app.py
   ```