Project Documentation: OCR Summarizer

1. Summary

The OCR Summarizer is a machine learning-integrated web application designed to digitize and condense static visual content. By combining Optical Character Recognition (OCR) with Natural Language Processing (NLP), the system extracts text from images and generates concise, abstractive summaries. The application features a minimalist web interface powered by Flask and utilizes the Facebook BART Large CNN model for high-quality text summarization.

2. Key Features

- **Optical Character Recognition (OCR):** Utilizes Tesseract technology to accurately extract raw text from various image formats (JPG, PNG), converting pixel data into machine-readable strings.
- **AI-Powered Summarization:** Implements a pre-trained Transformer model (BART) to analyze extracted text and generate coherent summaries, preserving key information while reducing word count.
- **Smart Preprocessing:** Includes an automated text cleaning pipeline that removes formatting noise (excessive newlines, form feeds) to ensure optimal input for the summarization model.
- **Portable Configuration:** The system is designed to detect and utilize a local, portable version of Tesseract, eliminating complex system-wide installation requirements for deployment.

3. System Architecture

3.1 Technical Stack

- **Language:** Python 3.8+
- **Web Framework:** Flask
- **OCR Engine:** Tesseract (via pytesseract)
- **Machine Learning:** Hugging Face Transformers, PyTorch
- **Image Processing:** Pillow (PIL)
- **Frontend:** HTML5, CSS3, JavaScript

3.2 Directory Structure

The project is organized as follows:

- **app.py:** The core Flask application entry point. Handles file uploads, OCR execution, model loading, and result rendering.
- **templates/:** Houses the index.html interface which includes the embedded CSS styling and JavaScript logic.

- **static/uploads/:** The destination folder where user-uploaded images are temporarily stored for processing.
- **requirements.txt:** Lists all necessary Python dependencies including transformers, torch, and pytesseract.
- **portable_tesseract_ocr/:** Contains the portable Tesseract binaries and data, enabling the application to execute OCR tasks without requiring a global system installation.

4. Installation & Setup Guide

Follow these steps to set up the development environment on a Windows machine.

Step 1: Configure PowerShell Permissions

To allow the activation of virtual environments, the execution policy must be adjusted for the current process.

Set-ExecutionPolicy Unrestricted -Scope Process

Step 2: Create Virtual Environment

Initialize a new Python virtual environment to isolate dependencies.

python -m venv venv

Step 3: Activate Environment

Activate the virtual environment to begin installing packages.

.\venv\Scripts\activate

Step 4: Install Dependencies

Install all required libraries listed in the requirements file.

pip install -r requirements.txt

Step 5: Tesseract Configuration

Ensure Tesseract is installed. You may either:

1. Check portable version in the folder named portable_tesseract_ocr in the project directory.
2. Or, ensure Tesseract is installed globally on your system PATH.

Step 6: Launch Application

Start the Flask server. *(Note: If running for the first time, the application will download the BART model (~1.6GB), which may take a few moments).*

python app.py

Step 7: Access Interface

Open your preferred web browser and navigate to the local server address:

http://localhost:5000

5. User Manual

How to Generate a Summary

1. **Select Image:** Click the dashed "Upload Zone" area to open the file explorer, or drag and drop an image file directly onto the box.
2. **Verify Selection:** The filename will appear below the upload zone to confirm your selection.
3. **Process:** Click the "Upload & Summarize" button.
4. **View Results:** The application will reload to display three distinct panels:
    - **Image:** A preview of the file you uploaded.
    - **Extracted Text:** The raw text captured by the OCR engine.
    - **Summary:** The AI-generated summary of the content.

6. Technical Specifications

6.1 Data Pipeline

The system processes user inputs through a sequential pipeline:

- **File Handling:** Images are saved using uuid generation to ensure unique filenames and prevent overwrites.
- **Text Cleaning:** Raw OCR output is processed using Regex (re.sub(r"[\n\f]+", " ", ocr_text)) to flatten the text structure before tokenization.

6.2 Machine Learning Model

- **Architecture:** Bart-Large-CNN (Facebook/Meta)
- **Task:** Abstractive Summarization
- **Configuration:**
    - **Max Input Length:** 1024 tokens
    - **Min Summary Length:** 20 tokens
    - **Sampling:** Disabled (do_sample=False) for deterministic outputs.
- **Tokenizer:** BartTokenizer is used to encode text into tensor format compatible with the PyTorch backend.