Image and Text Analyzer with Ollama

Author – Prashant Saxena

Github - https://github.com/p3rcyshots

# Introduction

* **Purpose:** This Python script, image\_analyzer.py, is a versatile tool designed to analyze images and text documents within a specified folder. It leverages the power of large language models (LLMs) through the Ollama framework to extract meaningful information, mirror document layouts, and provide both terminal output and structured document generation.
* **Target Audience:** This documentation is intended for developers, data scientists, and users interested in understanding, deploying, and customizing the image and text analysis capabilities of the script.
* **Key Features:**
  + Intelligent image analysis and document mirroring.
  + Automated image type detection (receipt, handwritten note, general document).
  + Flexible output options: terminal display and DOCX document generation.
  + Comprehensive error handling and logging for robust operation.
  + Customizable color-coded terminal output for enhanced user experience.
  + Progress indicators for long-running operations.
  + Command-line argument support for easy configuration.

# Components

* + **User:** Interacts with the script through the command-line interface (CLI).
  + **Command Line Interface:** Provides input to the script (arguments, prompts).
  + **image\_analyzer.py:** The main Python script containing the core logic.
  + **folder.txt:** A configuration file storing the path to the target folder.
  + **Ollama Model:** The LLM responsible for image and text analysis.
  + **Analysis Results:** The extracted information or mirrored document representation.
  + **Output Selection Logic:** Determines whether to display results on the terminal or generate a DOCX document (or both).
  + **Terminal Output:** Color-coded display of analysis results and script status.
  + **DOCX Document:** Structured document created using the python-docx library.
  + **analysis\_document.docx / image\_name\_mirror.docx:** Output DOCX files.

# Workflow

1. The user invokes the script with command-line arguments, specifying the Ollama model to use.
2. The script reads the target folder path from folder.txt.
3. The script analyzes files (images, documents) within the folder, determining image types when necessary.
4. The Ollama model is used to extract information or mirror document content.
5. Based on user prompts and script logic, analysis results are displayed on the terminal and/or written to DOCX files.

# Installation and Setup

* **Python Environment:**
  + Ensure you have Python 3.7 or higher installed.
  + It's highly recommended to use a virtual environment to manage dependencies.
* **Dependencies:** The script relies on the following Python packages:
  + ollama
  + python-docx
  + Pillow
  + PyPDF2
  + tqdm
* **Installation Instructions:**
  + **Create a virtual environment (optional but recommended):**
    1. python3 -m venv venv
    2. source venv/bin/activate # On Linux/macOS
    3. venv\Scripts\activate # On Windows
  + **Install the required packages:**

pip install ollama python-docx Pillow PyPDF2 tqdm

* **Ollama Setup:**
  + - Install Ollama from [https://ollama.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Follama.com%2F).
    - Ensure Ollama is running.
    - Pull the desired LLM model using the Ollama CLI:
    - ollama pull <model\_name>

Replace <model\_name> with the name of the Ollama model you want to use (e.g., llama2, mistral, codellama).

* **Configuration:**
  + **Create folder.txt:**
    - Create a file named folder.txt in the same directory as image\_analyzer.py.
    - Put the absolute path to the folder you want to analyze on the first line of this file (e.g., /path/to/your/folder or C:\path\to\your\folder).
  + **Place Images and Documents:** Put the images, PDFs, DOCX files, and other documents you want to analyze in the folder specified in folder.txt.

**4. Usage**

* **Command-Line Arguments:**
  + -m or --model: Required. Specifies the Ollama model to use.

Example:

python image\_analyzer.py -m llama2

* **Interactive Prompts:** Once the script is running, it will display the "Analyzer: " prompt. You can enter commands to:
  + analyze the images in the folder, and translate the content here with field and value of the field: Analyzes images and prints field-value pairs to the terminal.
  + create mirror document of the image: Creates a DOCX document that mirrors the content and layout of each image in the folder.
  + analyze files and create a document: Analyzes all files in the folder (images and documents) and creates a DOCX document with the analysis results.
  + Enter arbitrary text to chat with the Ollama model.
  + exit: Exits the script gracefully.

# Code Structure and Functionality

* **Core Functions:**
  + get\_current\_datetime(): Returns the current date and time as a formatted string.
  + check\_folder\_exists(folder\_path): Checks if a folder exists.
  + count\_images\_in\_folder(folder\_path): Counts the number of image files in a folder.
  + read\_text\_from\_pdf(pdf\_path): Extracts text from a PDF file.
  + read\_text\_from\_docx(docx\_path): Extracts text from a DOCX file.
  + get\_file\_contents(file\_path): Reads content from a file based on its extension.
  + analyze\_image\_with\_ollama(image\_path, ollama\_model, image\_type="general"): Analyzes an image using the specified Ollama model, adapting to image type (receipt, handwritten, general).
  + determine\_image\_type(image\_path, ollama\_model): Determines the image type using the Ollama model.
  + create\_mirror\_document\_from\_image(image\_path, ollama\_model, output\_path): Creates a DOCX document that mirrors the content of an image.
  + create\_docx\_from\_analysis(analysis\_results, output\_path): Creates a DOCX document from analysis results.
* **main() Function:**
  + Parses command-line arguments.
  + Handles Ctrl+C interrupts.
  + Reads the folder path from folder.txt.
  + Enters the main loop, processing user prompts and invoking analysis functions.
* **Color Coding:**
  + Uses ANSI escape codes to color the terminal output (light green for date/time, pink for LLM connection, yellow for folder information, light blue for everything else).

# Image Type Detection and Processing

* **Image Type Classification:**
  + The determine\_image\_type function sends a prompt to the Ollama model to classify the image as either "receipt", "handwritten", or "general".
  + The model's response is used to select the appropriate prompt in the analyze\_image\_with\_ollama function.
* **Prompt Engineering:**
  + The prompts used in analyze\_image\_with\_ollama are carefully crafted to guide the LLM to produce the desired output for each image type.
  + Receipts: Emphasize field-value extraction.
  + Handwritten Notes: Focus on accurate transcription.
  + General Images: Encourage content mirroring and layout preservation.

# Error Handling and Logging

* **try...except Blocks:** Extensive use of try...except blocks to catch potential exceptions and prevent the script from crashing.
* **Logging:** The logging module is used for informative error reporting and debugging. logging.info, logging.warning, and logging.error are used appropriately.
* **traceback.format\_exc():** Used to log the full stack trace when an exception occurs, providing detailed debugging information.
* **Signal Handling:** Uses signal.signal to catch SIGINT (Ctrl+C) and exit gracefully.

# Test Cases

* **Unit Tests (Conceptual):** Due to the reliance on external APIs (Ollama), traditional unit tests are challenging. Instead, focus on integration tests and validation of outputs. You could mock the Ollama calls for more isolated testing.
* **Integration Tests:**
  1. **Valid Folder Path:** Verify that the script correctly analyzes images and documents in a folder specified in folder.txt.
  2. **Invalid Folder Path:** Ensure the script handles the case where the folder path in folder.txt is invalid or does not exist.
  3. **Empty Folder:** Test with an empty folder to confirm that the script handles this scenario gracefully.
  4. **Image Analysis:**
     + Receipt: Verify that the script extracts field-value pairs from receipt images.
     + Handwritten Note: Ensure that the script transcribes handwritten notes accurately.
     + General Image: Confirm that the script mirrors the content and layout of general images.
  5. **Document Analysis:** Test that the script extracts text from PDF and DOCX files.
  6. **DOCX Generation:** Verify that the script creates DOCX documents with the correct content and formatting.
  7. **Error Handling:** Test that the script handles errors such as invalid image files, network connectivity issues, and Ollama API errors.
  8. **Ctrl+C Interrupt:** Confirm that the script exits gracefully when Ctrl+C is pressed.
* **Example Test Folder Structure:**

test\_folder/

receipt.jpg (Receipt image)

handwritten.png (Handwritten note image)

general.jpg (General image)

document.pdf (PDF document)

document.docx (DOCX document)

empty\_folder/ (Empty folder)

# Test Cases — Detailed

* **Approach:** Due to the reliance on external APIs (Ollama), formal unit tests are challenging. Focus on practical integration tests with clear setup steps, commands, expected input/output, and validation criteria. These tests will ensure the script functions correctly within a real-world environment.
* **Test Environment Setup:**
  1. **Create a dedicated test folder:**

mkdir test\_analyzer\_folder

cd test\_analyzer\_folder

* 1. **Populate the test folder:** Download or create the following test files and place them in the test\_analyzer\_folder directory:
     + receipt.jpg: A clear image of a typical receipt with recognizable field-value pairs (e.g., "Total: $25.00", "Date: 2024-03-27").
     + handwritten.png: An image of a handwritten note with legible text.
     + general.jpg: A photograph or other general image containing text.
     + document.pdf: A PDF document with several paragraphs of text.
     + document.docx: A DOCX document with several paragraphs of text.
     + empty\_file.txt: An empty text file.
  2. **Create a folder.txt file:** In the same directory as image\_analyzer.py, create a folder.txt file containing the *absolute* path to your test\_analyzer\_folder. For example:

/path/to/your/project/test\_analyzer\_folder # Linux/macOS

C:\path\to\your\project\test\_analyzer\_folder # Windows

## TC-1: Valid Folder Path (Image Analysis)

* + - **Purpose:** Verify the script analyzes images in a folder with a valid path.
    - **Command:**

python image\_analyzer.py -m llama2

* + - **Input (at the "Analyzer:" prompt):**

analyze the images in the folder, and translate the content here with field and value of the field

* + - **Expected Output:**
      * Colored terminal output showing the analysis of each image.
      * Receipt: Should display key-value pairs extracted (e.g., "Total: $25.00", "Date: 2024-03-27").
      * Handwritten: Should transcribe the note's text.
      * General: Should provide a description with key details.
    - **Validation:**
      * Verify that the correct Ollama model is loaded (check the initial script output).
      * Ensure that there are no errors in the terminal output.
      * Assess the accuracy of the extracted text and the descriptions for each image type.

## TC-2: Valid Folder Path (Create Mirror Document)

* + 1. **Purpose:** Verify the script creates DOCX mirror documents.
    2. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**

create mirror document of the image

* + 1. **Expected Output:**
       - The script should create a DOCX file for each image in the test\_analyzer\_folder named image\_name\_mirror.docx.
       - Colored terminal output indicating that the DOCX files have been saved.
    2. **Validation:**
       - Verify that receipt\_mirror.docx, handwritten\_mirror.docx, and general\_mirror.docx are created in the test\_analyzer\_folder.
       - Open each DOCX file and check that the content accurately mirrors the corresponding image (field-value pairs for receipts, transcription for handwritten notes, general description).

## TC-3: Valid Folder Path (File Analysis and Document Creation)

* + 1. **Purpose:** Verify the script analyzes images and files and creates a single DOCX document.
    2. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**

analyze files and create a document

* + 1. **Expected Output:**
       - The script should create a DOCX file named analysis\_document.docx in the same directory as image\_analyzer.py.
       - Colored terminal output indicating that the DOCX file has been saved.
    2. **Validation:**
       - Verify that analysis\_document.docx is created.
       - Open the DOCX file and check that it contains:
         * Analysis of receipt.jpg (field-value pairs).
         * Analysis of handwritten.png (transcription).
         * Analysis of general.jpg (general description).
         * Text from document.pdf and document.docx.
         * Empty Content in "empty\_file.txt".

## TC-4: Invalid Folder Path

* + 1. **Purpose:** Verify the script handles an invalid folder path gracefully.
    2. **Setup:**
       - Modify the folder.txt file to contain a non-existent folder path.
    3. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Expected Output:**
       - An error message in the terminal (colored in yellow) indicating that the folder does not exist.
       - The script should exit.
    2. **Validation:**
       - Confirm that the error message is displayed.
       - Verify that the script exits without further processing or errors.

## TC-5: Empty Folder

* + 1. **Purpose:** Verify the script handles an empty folder.
    2. **Setup:**
       - Modify folder.txt to point to an empty folder.
    3. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**

analyze files and create a document

* + 1. **Expected Output:**
       - A message in the terminal indicating that no analyzable files were found.
       - The script should create an analysis\_document.docx file that has a title named as "Analysis Result" and no other subheading or content.
    2. **Validation:**
       - Confirm the terminal message.
       - Check that the DOCX file is created but contains only the header.

## TC-6: Ctrl+C Interrupt

* + 1. **Purpose:** Verify the script exits gracefully when interrupted.
    2. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**
       - Press Ctrl+C while the script is running (e.g., during image analysis).
    2. **Expected Output:**
       - A message in the terminal (colored) indicating that Ctrl+C was detected and the script is exiting.
    3. **Validation:**
       - Confirm that the exit message is displayed and that the script terminates promptly.

## TC-7: Hand-written note with multiple languages

* + 1. **Purpose:** Verify that handwritten notes with multi-languages are detected.
    2. **Setup:** create an image with handwritten notes with english and another language.
    3. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**

analyze the images in the folder, and translate the content here with field and value of the field

* + 1. **Expected Output:**
       - Analysis of handwritten notes showing all languages
    2. **Validation:**
       - Check the output and verify whether languages are correctly analyzed

## TC-8: Check the name if a file exists while mirroring

* + 1. **Purpose:** Verify that the numbering is done correctly.
    2. **Command:**

python image\_analyzer.py -m llama2

* + 1. **Input:**

create mirror document of the image

* + 1. **Expected Output:**
       - New numbered file is created. The numbering continues.
    2. **Validation:**
       - Check whether new file is created

# Exception Handling

The code incorporates comprehensive exception handling to ensure robust operation. Here are some of the key exception handling scenarios:

* **File Not Found:**
  + Handles FileNotFoundError when folder.txt is missing.
  + Handles FileNotFoundError when image files are not found.
* **Invalid Folder Path:**
  + Checks if the folder path specified in folder.txt is a valid directory.
* **Ollama API Errors:**
  + Handles exceptions during Ollama API calls (e.g., network connectivity issues, invalid model name).
* **File Reading Errors:**
  + Handles exceptions when reading PDF, DOCX, and text files.
* **Image Processing Errors:**
  + Handles exceptions during image analysis.
* **DOCX Generation Errors:**
  + Handles exceptions during DOCX document creation and saving.
* **Ctrl+C Interrupt:**
  + Gracefully handles Ctrl+C interrupts, allowing the script to exit without errors.
* **Unexpected Errors:**
  + A generic except Exception as e block catches any unexpected errors and logs the full stack trace for debugging.

# Use Cases

* **Automated Document Processing:** Extract key information from invoices, receipts, and other documents.
* **Handwriting Recognition:** Transcribe handwritten notes and convert them into digital text.
* **Content Mirroring:** Create digital copies of documents that preserve their original layout and formatting.
* **Image Analysis:** Analyze images for specific objects, scenes, or text.
* **Custom Document Generation:** Generate DOCX documents based on extracted information and user-defined templates.
* **Chatbot Integration:** Integrate the script with a chatbot to provide image and text analysis capabilities to users.

# Future Enhancements

* **OCR Integration:** Integrate dedicated OCR libraries for enhanced text extraction from images.
* **Table Detection:** Add logic to specifically detect and extract tabular data from images.
* **Template Matching:** Implement template matching to identify document types and improve analysis accuracy.
* **User Interface:** Develop a graphical user interface (GUI) for easier interaction.
* **Cloud Deployment:** Deploy the script to a cloud platform for scalable processing.
* **More Sophisticated image detection with ML classifiers**: Enhance image type detection by using machine learning classifiers (trained on receipt, general, handwritten images) for improved accuracy.
* **Output in other formats** Add the option to have the analysis in other formats like JSON, XML, or CSV
* **Configurable Colors:** Allow users to configure the color scheme of the terminal output.

# Disclaimer

This script is provided as-is, without warranty of any kind. The accuracy of the analysis depends on the quality of the images and documents being processed, as well as the capabilities of the Ollama model being used. Always review the output carefully and verify its accuracy.

This documentation provides a thorough overview of the image\_analyzer.py script, covering its purpose, architecture, installation, usage, code structure, error handling, test cases, and potential future enhancements. It should serve as a valuable resource for developers and users seeking to understand and customize the script for their specific needs. Remember to always test the outputs to make sure they are accurate