**PROJECT: MVA DOC READER USING ML**

**GOAL:**

The primary goal of this project is to process scanned documents and generate a well-designed JSON object containing key-value data using machine learning techniques. The solution integrates image preprocessing, text detection, text recognition, and entity extraction with custom NLP models.

As of now the developed model can effectively identify 10 fields from any document if it is scanned well.

1. Document Name

2. Document Date

3. Applicant Name

4. Date of Birth

5. Driver’s License Number

6. Address (only if it is labelled with Address)

7. Conviction Date

8. Citation Date

9. Conviction Reason

10. NY Ref Id

**Tools & technologies:**

Python

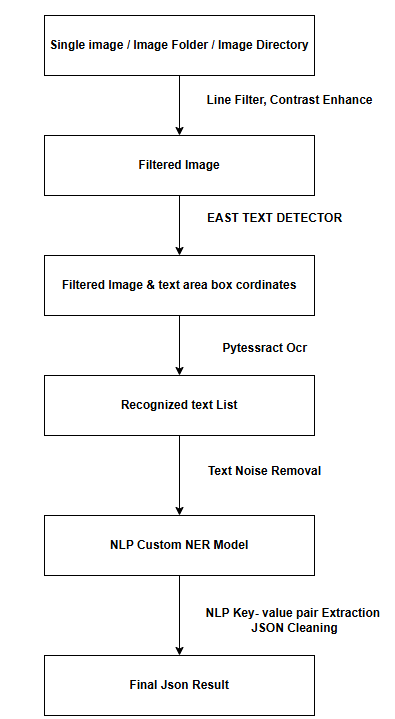
IDE (vs code)

NLP Spacy NER

EAST text Detector (Deep learning model)

Tesseract ocr

1. **WORKFLOW DIAGRAM:**



#### ****2. Project Directory****

* **config/**: Contains configuration files.
  + config.toml: Configuration parameters for the project.
* **logs/**: Stores log files generated during processing.
* **src/**: Core implementation code.
  + image\_filter.py: Handles image preprocessing (e.g., filtering, enhancing).
  + textdetector.py: Implements text detection and bounding box creation.
  + text\_recognition.py: Recognizes text within bounding boxes using Tesseract OCR.
  + key\_value\_extraction.py: Extracts key-value entities using SpaCy and other NLP techniques.
  + image\_processor.py: Orchestrates the image processing pipeline.
  + json\_cleaning.py: Cleans and formats the JSON output.
  + logger.py: Implements logging functionality.
* **main.py**: Entry point of the application.
* **requirements.txt**: Lists dependencies required for the project.
* **models/**: Contains pre-trained models (e.g., SpaCy models, EAST text detector).
* **config.ini**: Contains runtime configuration settings.
* **app.log**: Log file for tracking application execution.
* **web.config**: Configuration file for hosting on IIS.

**3.1. Image Preprocessing**

**Goal**

Enhance the input image to improve the accuracy of text detection by applying filtering techniques using OpenCV.

**Steps**

1. **Load the Image**
   1. Accepts input as a single file, a folder of images, or from a directory path.
   2. Reads the image using OpenCV and prepares it for processing.
2. **Apply OpenCV Filtering Techniques**
   1. **Contrast Enhancement**: Converts the image to grayscale and applies thresholding to separate text from the background.
   2. **Line Filtering**: Detects vertical and horizontal lines using morphological transformations.
      1. **Vertical Line Detection**:
         1. Uses vertical structuring elements to isolate vertical lines based on the image's height.
      2. **Horizontal Line Detection**:
         1. Applies horizontal structuring elements to detect horizontal lines based on the image's width.
   3. Combines vertical and horizontal lines for a unified mask.
3. **Remove Lines**
   1. Removes detected lines from the image using inpainting techniques.
   2. Inpainting fills the regions of the lines with surrounding pixel information.
4. **Save or Pass the Processed Image**
   1. The enhanced image is saved or passed to the next step in the pipeline for text detection.

**3.2. Text Detection & Box Bounding**

**Goal**

Detect dense text areas within an image and draw bounding boxes around them. This will enable the identification of regions containing text for further processing, such as text recognition and entity extraction.

**Steps**

1. **Use the EAST Text Detector:**
   * Utilize the pre-trained EAST (Efficient and Accurate Scene Text Detector) model to localize text in the image.
   * Perform forward propagation of the image through the model to obtain scores and geometry data for potential text regions.
2. **Non-Max Suppression:**
   * Eliminate overlapping bounding boxes to retain the most relevant ones by applying non-max suppression (NMS). This helps in refining the detection results by removing redundant boxes.
3. **Bounding Box Processing:**
   * Adjust and dilate the detected boxes to ensure no significant text area is missed.
   * Use morphological operations to refine the bounding box boundaries further.
4. **Draw Bounding Boxes:**
   * Apply the detected bounding boxes to the original image, highlighting the text regions.
   * The output is the image with drawn boxes and the coordinates of the detected bounding boxes.

**Output**

* **Image with Bounding Boxes:** The original image with rectangular boxes drawn around detected text areas.
* **Bounding Box Coordinates:** A list of coordinates for the detected bounding boxes, each in the format (startX, startY, endX, endY).

**3.3. Text Recognition**

**Goal:**  
To recognize and extract text from the regions of interest (ROIs) defined by bounding boxes on the preprocessed image using Tesseract OCR.

**Steps:**

1. **Input Preparation**:
   * The input includes:
     + The preprocessed image.
     + Bounding box coordinates of text-dense areas.
   * Bounding boxes are sorted by their Y-coordinates and then by X-coordinates to maintain a natural reading order.
2. **Region Extraction**:
   * Each bounding box is used to extract a region of interest (ROI) from the image.
3. **Text Recognition**:
   * Apply Tesseract OCR to each ROI using the --psm 6 configuration (assumes a single block of text with uniform alignment).
   * Extracted text is cleaned by removing unwanted newline characters and trailing spaces.
4. **Parallel Processing**:
   * Leverage multithreading with ThreadPoolExecutor to perform OCR on multiple bounding boxes simultaneously, improving efficiency.
5. **Output Generation**:
   * Filter out empty results to ensure meaningful text data.
   * Return a list of recognized text, maintaining the reading order.

**3.4. NLP SpaCy Model Development**

**Goal**

To train a custom SpaCy NER model to recognize specific entities (DOC\_NAME, DOC\_DATE…..) from the detected text in the scanned documents.

**Introduction**

SpaCy is a library for building Natural Language Processing (NLP) applications. One of its prominent features is Named Entity Recognition (NER), which identifies and classifies entities in text into predefined categories.

**Steps**

1. **Prepare Training Data**:
   * Annotate the text data with entity labels (APPLICATION, LICENSE\_NO, etc.).
   * Save the annotated data in JSON format. For example:
2. **Load and Preprocess Data**:
   * Load the training data from a JSON file and preprocess it into SpaCy's DocBin format.
3. **Train the Model**:
   * Create a blank NLP pipeline.
   * Add the annotated data to the pipeline using DocBin.
   * Use the spacy init config command to generate a configuration file for training.
   * Train the model using the spacy train command.
   * Specify the training and development datasets in .spacy format.
   * Train the NER model using SpaCy commands.

!python -m spacy init config --lang en --pipeline ner config.cfg --force

!python -m spacy train config.cfg --output ./Model\_123456789\_T1 --paths.train ./DC\_train\_T.spacy --paths.dev ./DC\_test\_T.spacy

1. **Save the Trained Model**:
   * Save the trained SpaCy model in the models/ directory for use in the entity extraction step.

**3.5. Key- Value Extraction with Spacy**

**Goal**: The goal of the **Key-Value Extraction** step is to accurately recognize and extract meaningful key-value pairs from the text obtained in the previous stages (i.e., after text recognition). The extracted data will represent specific document entities such as APPLICANT\_NAME, LICENSE\_NO, DATE OF BIRTH, DOCUMENT DATE, etc. The extracted key-value pairs will then be cleaned, formatted, and converted into a structured JSON object.

1. **Noise Removal**
   * **Objective**: Eliminate unwanted text noise that may distort the OCR output.
   * **Implementation**:
     + Use the CleanText class to remove specific noise patterns from recognized text.
     + Text noise includes unwanted words and characters that appear due to OCR artifacts (e.g., DRIVER, MIDDIE, special characters).
     + The noise patterns to remove are defined in the CLEANING\_PATTERNS dictionary.
2. **NLP Key-Value Extraction**
   * **Objective**: Extract key-value pairs from cleaned text using the custom SpaCy NER model.
   * **Implementation**:
     + Pass the cleaned text to the NLPKeyValueExtraction class.
     + Use the SpaCy model to recognize named entities.
     + Map recognized entities to predefined labels (e.g., APPLICATION, LICENSE\_NO).
     + Aggregate key-value pairs while ensuring unique entries.
3. **Data Cleaning**
   * **Objective**: Clean the extracted JSON data to remove unwanted patterns and special characters.
   * **Implementation**:
     + Use the clean\_ocr\_json() method to clean up the key-value pairs.
     + Apply predefined cleaning patterns stored in the CLEANING\_PATTERNS dictionary.
     + Remove specific patterns (e.g., names, license identifiers) to ensure proper formatting and readability.
4. **Generate Clean JSON**
   * Format the key-value pairs into a well-structured JSON object.
   * Return the cleaned and formatted JSON to be stored or outputted according to downstream application requirements.

**4.Generating a Requirements File in Python**

A **requirements.txt** file lists all the dependencies (Python packages) required for your project. This file allows others to set up the same environment easily by installing the necessary packages with a single command.

**Steps to Generate a Requirements File**

**Option 1: Using pipreqs**

The pipreqs package automatically generates a requirements.txt file based on the imported libraries in your project.

1. **Install pipreqs**:

pip install pipreqs

1. **Generate the requirements.txt File**:
   * Navigate to your project directory in the terminal.
   * Run the following command:

pipreqs . --force

* + - The . specifies the current directory.
    - --force overwrites an existing requirements.txt file.

1. **Result**:
   * A requirements.txt file will be generated in your project directory containing all imported dependencies.

**Option 2: Using pip freeze**

pip freeze captures all installed packages in your current Python environment and their versions.

1. **Generate the requirements.txt File**:
   * Run the following command in your virtual environment:

pip freeze > requirements.txt

1. **Customize the requirements.txt File (Optional)**:
   * Open the file in a text editor.
   * Remove unnecessary packages if they are not used in your project.

**Using the Requirements File**

1. To install the dependencies listed in requirements.txt in a new environment, use the following command:

pip install -r requirements.txt

1. Ensure that the environment matches the Python version used to create the file to avoid compatibility issues.

### 5.Internet Information Services (IIS) Hosting Documentation

**Overview**

IIS (Internet Information Services) is Microsoft’s web server used for hosting, deploying, and managing web applications. IIS allows internet users to access web pages and supports both static content (e.g., HTML, CSS, JavaScript) and dynamic content generated via web applications like Flask.

Key Components:

1. **Handler Mappings**: Manage URL handlers that process requests for specific file types.
2. **Fast CGI**: A protocol to reduce the overhead between web servers and CGI applications, enabling faster request handling**Hosting**

**Flask Applications on IIS**

Flask applications can be hosted on IIS to serve HTML, CSS, and APIs with Flask's data-passing capabilities. Below are the steps and troubleshooting notes for deploying a Flask app in IIS**.**

**Preparation**

1. **Set up your Flask Project**:
   * Ensure all .py files and dependencies are in a project directory.
   * Add a web.config file to the root directory.
2. **Set up a Python Virtual Environment**:
   * Create a virtual environment (preferably with Anaconda or venv).
   * Install the required dependencies using pip install -r requirements.txt.
   * Verify your Flask app runs correctly by testing locally.

**Steps to Host Flask Application on IIS**

**Step 1: Create a New Website in IIS**

1. Open **IIS Manager**:
   * Right-click **Sites** > **Add Website**.
2. Enter the following:
   * **Site Name**: A descriptive name for your site.
   * **Physical Path**: Directory path where the Flask project is located.
   * **Port**: Assign a unique port number.
3. Click **OK** to create the site.

**Step 2: Configure Handler Mappings**

1. Install **wfastcgi**:
   * Run the command:

pip install wfastcgi

1. Enable FastCGI for Flask using:

wfastcgi-enable

* + This will output the script processor path (e.g., c:\projects\env\scripts\python.exe|c:\projects\env\lib\site-packages\wfastcgi.py).

1. Add the script processor path to **Handler Mappings**:
   * Open IIS > Navigate to your website > **Handler Mappings** > **Add Module Mapping**.
   * Enter the following:
     + **Request Path**: \*.
     + **Module**: FastCgiModule.
     + **Executable**: Paste the script processor path from the previous step.
     + **Name**: Any descriptive name (e.g., FlaskHandler).
   * Click **Request Restrictions**:
     + Under **Access**, select **Execute**.
   * Click **OK** to save the handler.

**Step 3: Configure FastCGI Settings**

1. In IIS, go to **FastCGI Settings**.
2. Verify the script processor path is added correctly.
3. Edit the FastCGI settings if needed (e.g., modify timeouts for long-running requests).

**Step 4: Assign Permissions**

Flask applications require appropriate permissions to access files and execute scripts.

1. **Grant Folder Permissions**:
   * Right-click your project directory > **Properties** > **Security** > **Edit**.
   * Click **Add** > **Object Types** > Check **All** > OK.
   * Click **Locations** > Select the root machine > OK.
   * Search for **IUSR** and **IIS\_IUSRS**, then click **OK**.
   * Assign **Modify** and **Full Control** permissions.
   * Apply changes.
2. **Grant Python Executable Permissions**:
   * If using Anaconda, ensure **conda.exe** and Python executables have similar permissions.

**Step 5: Update web.config File**

1. Create a web.config file in your project directory.
2. Include the following content, adjusting paths as needed:

<configuration>

<system.webServer>

<handlers>

<add name="Python FastCGI" path="\*" verb="\*" modules="FastCgiModule" scriptProcessor="C:\path\_to\_python\python.exe|C:\path\_to\_wfastcgi\wfastcgi.py" resourceType="Unspecified" />

</handlers>

</system.webServer>

</configuration>

**Step 6: Test Your Application**

1. Open your web browser and navigate to http://localhost:<port> or your assigned domain.
2. If you encounter errors, check the **Event Viewer** logs for debugging.

**Troubleshooting**

**Error: HTTP Error 500.0 - Internal Server Error**

* **Cause**: The FastCGI process exited unexpectedly.
* **Solution**:
  1. Verify IUSR and IIS\_IUSRS permissions for your project directory.
  2. Ensure the Python executable and dependencies are accessible.
  3. Restart IIS using the following command:

Iisreset

s

**Other Common Issues:**

* **Port Conflicts**: Check that the assigned port is not already in use.
* **Invalid Handler Path**: Ensure the script processor path matches the output of the wfastcgi-enable command.

**Reference Steps for IIS Hosting**

1. Create a project folder and ensure all .py files and dependencies (e.g., requirements.txt, web.config) are included.
2. Set up a virtual environment and install required dependencies.
3. Run and test your Python application locally.
4. Use the wfastcgi-enable command to configure FastCGI and copy the script processor path.
5. Add a new site in IIS with the project folder path and assign a port.
6. Configure handler mappings using the script processor path.
7. Grant appropriate permissions to IUSR and IIS\_IUSRS.
8. Restart IIS and test your hosted application.

**References**

* [wfastcgi GitHub Documentation](https://github.com/Microsoft/PTVS/wiki/Flask-Web-Project)
* YouTube Tutorial: [Hosting Flask App on IIS](https://youtu.be/En9vo7Ognm0)
* IIS Official Documentation: [FastCGI Configuration](https://learn.microsoft.com/en-us/iis/)

# **6. To Run Application**

**OPTION 1:**

1. Create a Virtual Environment by below command  
   py -m venv <name\_of\_env>
2. Activate the Virtual Environment:  
   .<name\_of\_env>\Scripts\activate
3. Install Dependencies: pip install -r requirements.txt

**OPTION 2:**

1. Create a Conda Environment: conda create -p <path\_to\_env> python==<version> -y
2. Activate the Conda Environment: conda activate <path\_to\_env>
3. Install Dependencies: pip install -r requirements.txt.

# **7. Hosting & Deployment**

**Hosting On IIS**

* 1. Configure web.config for IIS hosting
  2. Use the conda virtual env for runtime dependencies

# **8. Conclusion**

This project automates the extraction of structured data from scanned documents into JSON format using advanced image processing and NLP techniques. The modular design ensures scalability and maintainability.