**License Plate Detection & Recognition App**

**Project Overview**

The License Plate Detection & Recognition app is built using YOLOv8 for license plate detection and OCR (Optical Character Recognition) for text extraction. The objective was to build a robust solution that could detect license plates from images and videos, and extract the text on the plates. The app has been designed for both command-line interface (CLI) and user-friendly interaction through a Streamlit web app.

**Approach**

**1. Data Preparation**

The project began by preparing a custom dataset for license plate detection using **Roboflow** in YOLOv8 format. The dataset consisted of images with corresponding label files (.txt) containing bounding box information for license plates.

**2. Model Training**

The model was trained using YOLOv8, leveraging a combination of the custom dataset and pretrained weights for optimal performance. Two types of YOLOv8 models (yolov8n.pt and yolov8x.pt) were tested to balance accuracy and computational efficiency.

Key parameters for model training:

* **Image size:** 640x640 pixels
* **Confidence threshold:** 0.25
* **Intersection over Union (IoU) threshold:** 0.45

**3. License Plate Detection**

After training, the detect\_modified.py script was developed to handle license plate detection in both images and video files. The detection pipeline leverages the YOLOv8 model to identify plates in frames, after which EasyOCR is used for text extraction.

Key steps in detection:

* Load the YOLOv8 model weights.
* Run the detection pipeline on the provided source (image or video).
* Post-process the bounding box coordinates and confidence scores.
* Apply Optical Character Recognition (OCR) using **EasyOCR** to extract text from the detected license plates.

**4. Web Application (Streamlit)**

A **Streamlit** application was developed to provide a user-friendly interface. The app allows users to upload images or videos for real-time license plate detection. The extracted license plate text is displayed, along with the bounding boxes around detected plates.

* **File Upload:** Users can upload images or videos.
* **Detection Display:** The detected license plates and extracted text are displayed.
* **Configuration:** Users can modify the model's confidence threshold and image size via a configuration file.

**Challenges Faced**

1. **Dataset Size & Quality:** One of the key challenges was the relatively small dataset available for training. To improve the model's robustness, data augmentation techniques were applied, but achieving optimal performance in real-world, noisy environments remains an ongoing task.
2. **OCR Performance:** The OCR results were sensitive to the quality of the license plate images, especially when plates were blurry or distorted. Enhancing the post-processing of the detected regions for clearer OCR results is a future goal.
3. **Real-Time Processing:** For video streams, processing speed became a challenge. A balance between model size (YOLOv8n vs YOLOv8x) and real-time performance had to be achieved.
4. **Hosting and Dockerization:** Dockerizing the app for deployment introduced challenges related to container size and managing dependencies like torch and opencv-python. Hosting the app in a cloud environment added further complexity, but these were mitigated using Docker best practices and testing locally before deployment.

**Results**

The detection system worked effectively on clean and well-lit license plate images. It achieved accurate text extraction under optimal conditions, though performance dropped slightly on low-quality images and videos.

* **Detection Accuracy:** High accuracy on the custom dataset, with a slight reduction on real-world images with poor lighting or angled plates.
* **OCR Performance:** Moderate performance, with reliable extraction on clear images, but challenges on blurry or low-resolution plates.
* **Real-Time Video Detection:** Capable of processing video streams, but additional work is needed to optimize the frame rate for real-time detection.