🚗 License Plate Recognition Web Application

📝 Project Overview

This project focuses on developing a **License Plate Recognition (LPR)** system that allows users to upload vehicle images and automatically extract license plate numbers using **OCR (Optical Character Recognition)**. The application is built using **Flask** for the backend, **OpenCV** for image preprocessing, and **EasyOCR** for text extraction.

🎯 Objective & Motivation

In real-world scenarios like parking automation, traffic monitoring, and law enforcement, manual identification of vehicle plates is time-consuming and error-prone. This project was undertaken to:

- **Automate** the detection and recognition of license plates from vehicle images.
- Build an easy-to-use **web interface** for testing OCR capabilities on vehicle data.
- Explore the integration of **computer vision and OCR** in practical systems.

🔄 Project Flow

1. Image Upload

- The user uploads a vehicle image through the web interface.

2. Preprocessing

- The image is converted to grayscale.

- Bilateral filtering and Canny edge detection are applied to enhance features.
- Contour detection is used to identify rectangular regions likely to contain plates.

3. License Plate Detection

- Among the contours, the one with four corners (quadrilateral) is selected as the potential license plate region.
- A mask is created to isolate this region and crop it.

4. Text Extraction (OCR)

- The cropped license plate region is passed to **EasyOCR**.
- The extracted text is returned and displayed on the frontend.

5. Result Display

- The detected plate and text are shown on the web page along with the uploaded image.

🛠️ Tech Stack

- **Frontend:** HTML, CSS, Jinja2 (templating)
- **Backend:** Python, Flask
- **Computer Vision:** OpenCV, NumPy, imutils
- **OCR Engine:** EasyOCR (PyTorch-based)

📊 Performance

- Tested on a variety of vehicle images with different angles and lighting.
- Achieved **over 90% character-level OCR accuracy** on well-lit and front-facing plate images.
- Processing time per image: ~1–2 seconds on standard CPU.

⚙️ Challenges Faced

1. **Plate Detection Accuracy:**
    - Non-standard plate shapes, dirt, or shadows made detection tricky.
    - Contour-based detection sometimes failed on side-view images.
2. **OCR Errors:**
    - EasyOCR occasionally misread characters like `O` vs `0`, `B` vs `8`.
3. **Frontend Integration:**
    - Ensuring a smooth image upload + display pipeline with Jinja2 templates.

📚 What I Learned

- **Image preprocessing techniques** for edge detection, filtering, and contour approximation.
- How to integrate **OCR into web applications** using pre-trained models.
- Structuring a **modular Flask app** with static file serving and templated HTML views.
- Practical insights into **deployable computer vision pipelines**.

🔄 Future Enhancements

- Add **real-time webcam support** for live license plate scanning.
- Log detected plates with **timestamp and location**.
- Deploy app on **cloud platforms** like Render, Heroku, or AWS EC2.
- Use **YOLO or custom-trained models** for better plate localization.