End-to-End ALPR (Automated License Plate Recognition) Pipeline

This project is a full-stack web application that performs Automated License Plate Recognition using a custom-trained **YOLOv8** object detection model and **Tesseract-OCR**.

The user can upload an image of a car, and the application will send it to a Python Flask backend. The backend then runs a sophisticated ML/CV pipeline to detect the license plate, isolate it, and read the characters, sending the results back to the frontend.

Final Application

The web interface shows the user's uploaded image and the results of each step in the processing pipeline, including the final recognized text.

Tech Stack

- **Backend:** Python, Flask (for the web server/API)
- **Machine Learning:** PyTorch, Ultralytics YOLOv8 (for custom model training and plate detection)
- **Computer Vision:** OpenCV (for image processing, pre-processing, and perspective warping)
- **OCR:** Pytesseract (for reading the text from the isolated plate)
- **Frontend:** HTML, TailwindCSS, JavaScript (for the user interface and API communication)

Features

- **Custom-Trained ML Model:** The license plate detector is a YOLOv8 'nano' model custom-trained on a large, public dataset.
- **Full Pipeline Visualization:** The UI shows the result of every major step, from the original upload to the final text.
- **Client-Server Architecture:** A robust Flask server handles all heavy processing, keeping the frontend fast and light.
- **End-to-End Solution:** A complete project that goes from a raw image to a final text string (e.g., "CZCV565").

How to Set Up and Run This Project

This is an advanced project with two major parts. **You must train your own model first** before you can run the final web application.

Prerequisites

1. **Python 3.10+**: Make sure Python is installed and added to your PATH.
2. **Tesseract-OCR Engine**: This project requires the Tesseract program to be installed *on your computer* (not just the Python library).
    - **Download:** https://github.com/UB-Mannheim/tesseract/wiki
    - **Critical:** During installation, you **MUST** check the box to **"Add Tesseract to your system PATH."**

Part 1: Training Your Custom ML Model

1. **Clone the Repository:**

```
git clone [https://your-repo-link-here.git](https://your-repo-link-here.git)
cd alpr-project
```

2. **Create a Virtual Environment:**

```
python -m venv venv
```

3. **Activate the Environment:**
   - **Windows:** `.\venv\Scripts\activate`
   - **macOS/Linux:** `source venv/bin/activate`
4. **Install Python Dependencies:** (This will take several minutes as it installs PyTorch)

```
pip install -r requirements.txt
```

5. **Download the Dataset:**
   - This project is configured for the dataset at: https://www.kaggle.com/datasets/fareselmenshawii/large-license-plate-dataset
   - Download the `archive.zip` file.
   - Unzip the file and place the contents (the `images` and `labels` folders) into a new folder named `archive` inside your `alpr-project` directory.
   - Your project structure should look like this:

```
alpr-project/
│
├── archive/  <-- The dataset you just added
│   ├── images/
│   │   ├── train/
│   │   ├── val/
│   │   └── test/
│   └── labels/
│       ├── train/
│       ├── val/
│       └── test/
│
├── venv/
├── alpr_pipeline.py
├── app.py
├── index.html
├── dataset.yaml
├── train.py
└── requirements.txt
```

6. **Verify `dataset.yaml`:**
    - Open the `dataset.yaml` file.
    - Make sure the `path:` line correctly points to the **full, absolute path** of your `archive` folder.
    - (The file is already configured for the `images/train` and `labels/train` sub-folder structure).
7. **Run the Training!**
    - This is the most time-consuming step. It will take **many hours** on a CPU.

```
(venv) > python train.py
```

    - Let it run until it completes all 50 epochs. When finished, you will have your new, custom-trained model at: `runs/detect/train/weights/best.pt`

Part 2: Running the Final Web Application

Once your `best.pt` model file exists, you are ready to run the app.

1. **Verify Model Path:**
    - Open `alpr_pipeline.py`.
    - Ensure the `YOLO_MODEL_PATH` variable at the top points to your new `best.pt` file. (The default path `runs/detect/train/weights/best.pt` should already be correct).
2. **Run the Flask Server:**
    - In your terminal (with `(venv)` active), run:

```
(venv) > python app.py
```

    - You should see `* Running on http://127.0.0.1:5000`. This means your backend is live.
3. **Open the Frontend:**
    - In your file explorer, find and **double-click the `index.html` file**.
    - This will open the application in your web browser.
4. **Test It!**
    - Upload an image of a car and click "Run ML Pipeline."
    - You should see all four steps execute and the final text appear.

File Overview

- `app.py`: The Flask web server that handles API requests.
- `alpr_pipeline.py`: The core logic file. Loads the YOLO model and Tesseract to run the full pipeline on an image.
- `train.py`: The one-time script used to train the YOLOv8 model.

- `dataset.yaml`: The config file that tells `train.py` where to find the dataset.
- `index.html`: The frontend web page (UI).
- `requirements.txt`: List of all Python libraries needed for the project.
- `README.md`: This file!