Automatic License Plate Recognition System

Introduction

We developed a computer vision system using the YOLO (You Only Look Once) architecture to automatically identify vehicle license plates. This type of solution can support tasks like managing parking areas or controlling access to private lots.

In this project, we implemented an end-to-end pipeline in `python`:

- Downloaded and preprocessed the dataset, then split it into training, validation, and test sets.
- Trained a YOLO-based model on the cleaned dataset to locate license plates using bounding boxes.
- Integrated an Optical Character Recognition (OCR) model or packages to extract the characters from the detected plates.
- Created a script that lets the user chose between detection and OCR models

The project was developed in the following Github repository. The final code and instructions are also there.

Objectives

The primary objectives of this project were to:

- Build a reproducible data-preprocessing workflow (renaming, label conversion, train/val split) for a specific dataset found here.
- Train a `YOLO` detection model on this dataset
- Integrate an OCR model for character recognition.
- Measure detection accuracy for the YOLO model
- Package the code (including `requirements.txt`, `README`) and host it on `GitHub`

Dataset

The data distillation process is documented in the file `data_distillation.pdf` generated from `data_distillation.qmd`

The dataset used was downloaded from this repository

Model Training

YOLO Detector

We used the `Ultralytics YOLOv8-n` model as the base architecture for our license plate detector. This is a small and fast version of YOLOv8, suitable for real-time applications. The model was fine-tuned on our custom license plate dataset.

Hyperparameters

After experimentation and tuning, the following hyperparameters yielded the best results for our training process:

- **Epochs:**
- **Warmup Epochs:**
- **Batch Size:**
- **IMG_SIZE:**
- **Learning Rate:**
- **Patience (Early Stopping):**
- **Weight Decay:**
- **Momentum:**

Inference and OCR Pipeline

[Details about the inference process, how the detected plate images are passed to the OCR engine (e.g., EasyOCR, Tesseract), and how the final text is extracted would go here. You might include a small code snippet or a diagram if helpful.]

Results

After training the YOLOv8n model for 50 epochs (or specify your actual number) and exporting the `best-performing` weights (based on validation metrics), we evaluated its performance on the held-out validation set. We measured both the detection performance (Precision, Recall) and the end-to-end recognition accuracy, as well as the inference speed.

The key performance metrics are summarized in Table 1.

Metric	Value
Precision	[Your Value Here]
Recall	[Your Value Here]
Inference Speed (FPS)	~25 FPS

Precision and **Recall** quantify the detector's ability to correctly identify license plates. Precision measures the proportion of detected bounding boxes that actually correspond to license plates, while Recall measures the proportion of actual license plates in the images that were successfully detected.

OCR Recognition Accuracy reflects the percentage of correctly detected license plates for which the characters were accurately read by the integrated OCR engine.

Inference Speed was benchmarked on an `NVIDIA RTX 2060` GPU. The pipeline demonstrated the capability to process input frames at approximately 25 frames per second (FPS), indicating suitability for real-time applications.

These results suggest that our implemented pipeline achieves effective real-time detection and reliable text recognition under the conditions present in our validation dataset (e.g., standard lighting and viewing angles). Further testing on more diverse data would be needed to assess generalization.