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 model for image detection

We used the `Ultralytics YOLOv11` models as the base architecture for our license plate detector. The YOLO detection models were originaly trained on the COCO dataset. For details on datasets used, see the Dataset section.

We tried with multiple sizes of models, and finaly where able to train the biggest one (`x version`). The model was fine-tuned on our custom license plate dataset.

The base code used for the training can be found in `train_yolo.py`. We used Google Colab, as we needed GPU resources and there it's already set to go. CUDA installation was also performed in all of our machines, but we saw faster results which did not crash that often in a Google Colab.

Hyperparameters

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

- **Epochs:** 100
- **Warmup Epochs:** 2
- **Batch Size:** 16
- **Learning Rate:** 0.001 (Initial one)
- **Patience (Early Stopping):** 10
- **Weight Decay:** 0.0005 (regularization to avoid overfitting)
- **Momentum:** 0.937 (number seen to work well, optimizer momentum)

Results

- Training images (`train_batch0.jpg`, `train_batch1.jpg`, `train_batch2.jpg`) show successful detection of license plates.
- The model achieved accurate bounding box predictions for most test images.

After training the YOLO11 model we exported the `best-performing` weights and used it as a model for our plate-number detection workflow (models uploaded with lfs). Run `script.py` by following the instructions of the README in the repository to try the models.