Truck-Trailer-Counting Documentation

December 7, 2024

Contents

1 Introduction

The **Truck-Trailer-Counting** project provides an efficient and scalable solution for detecting and counting trucks in video footage. The system uses deep learning techniques, specifically YOLO-based object detection, for real-time applications. It is designed for use cases such as:

- Traffic flow analysis
- Logistics optimization
- Road safety monitoring

2 Hardware and Software Environment

2.1 Hardware Environment

- **CPU**: Intel Core i7-8700 @ 3.20GHz
- **GPU**: NVIDIA GeForce RTX 2070 SUPER
- **RAM**: 48 GB DDR4 @ 2666 MHz

2.2 Software Environment

- **OS**: Windows 11 Pro (64-bit)
- **Development Tools**: Jupyter Notebook 7.x
- **Framework**: PyTorch 2.x
- **Libraries**: OpenCV, Ultraytics, NumPy

3 Dataset Preparation

3.1 Structure

The dataset is organized into the following structure:

```
dataset/
   train/     # Training data
   val/       # Validation data
   test/      # Test data
```

3.2 Dataset Splitting

The dataset is split into training, validation, and test subsets:

- **Training Set**: 70% of the dataset
- **Validation Set**: 20% of the dataset
- **Test Set**: 10% of the dataset

A Python script automates the splitting process by shuffling the dataset and creating separate directories for each subset.

3.3 YAML Configuration

The 'data.yaml' file defines the dataset paths and class names. It specifies:

- Paths to the training and validation data.
- Number of classes (`nc`).
- Class names (e.g., `Truck`).

4 Data Augmentation

To improve model robustness, several data augmentation techniques are applied:

- **Scaling**: Randomly resize images and adjust bounding boxes accordingly.
- **Horizontal Flip**: Flip images horizontally with a 50% probability.
- **Brightness Adjustment**: Modify brightness to simulate different lighting conditions.

Each augmentation step generates new image and annotation pairs, increasing the diversity of the dataset.

5 Model Training

5.1 Model Overview

The model is based on YOLO (You Only Look Once), a state-of-the-art object detection algorithm. The pre-trained YOLOv8 model is fine-tuned on the prepared dataset to optimize its performance for truck detection.

5.2 Training Parameters

The training process involves:

- Defining the dataset configuration using the `data.yaml` file.
- Training the model for 199 epochs with a batch size of 37.
- Using GPU acceleration for faster training.

The trained model is saved for later use in real-time applications.

6 Real-Time Video Processing

The system processes video footage to detect and count trucks crossing predefined Region of Interest (ROI) lines.

6.1 Horizontal and Vertical Counting

- **Horizontal ROI Line**: Counts trucks crossing a line horizontally in the video frame.
- **Vertical ROI Line**: Counts trucks crossing a vertical line.

6.2 Steps in Video Processing

1. Open the video file and read each frame.
2. Apply the YOLO model to detect trucks in the frame.
3. Identify whether a detected truck crosses the defined ROI line.
4. Update the count if a truck is detected crossing the line.
5. Annotate the frame with bounding boxes and counts.
6. Display or save the processed video.

7 Results and Outputs

- **Bounding Boxes**: Visualized on detected trucks in real-time.
- **Truck Count**: Displayed on-screen and updated dynamically.
- **Annotated Video**: Saved for post-analysis.

8 Usage Guide

1. Install the necessary Python packages:

   ```
   pip install pandas numpy torch torchvision opencv-python ultralytics PIL glob2
   ```

2. Prepare the dataset and split it into subsets.
3. Fine-tune the YOLO model using the training data.
4. Process a video file by running:
5. Access the outputs, including the JSON results and annotated video.

9 Conclusion

This project demonstrates how deep learning can be applied effectively to real-time object detection and counting. Its modular design ensures easy scalability and customization for various use cases.