Digital Egypt Pioneers

Real-time Object Detection For Autonomous Vehicles

Group code: ONL2_XAIS2_S1

NHA / Skills Dyanmix

Submitted by:

Names
Aya Elmogy
Kareem Yasser
Aliaa Abobakr
Shiref Ashraf
Shehab Ahmed

Submitted to:

Eng. Zyad Mohammed

1 Introduction

This project aims to build a real-time object detection system for autonomous vehicles, identifying objects like cars, pedestrians, and cyclists to enhance safety and decision-making. The KITTI dataset, featuring annotated driving scenario images, was used to train a YOLO-based model. The pipeline includes data preparation, augmentation, model selection, training, evaluation, and deployment.

1.1 Project Objectives

- Convert KITTI dataset annotations to YOLO format.
- Apply data augmentation to improve model robustness.
- Train and evaluate YOLO models, comparing versions 8, 10, and 11.
- Log experiments with MLflow and export the final model to ONNX format.

2 Dataset

The KITTI Object Detection Benchmark dataset was used, with 1242×375 pixel images annotated for objects like cars, pedestrians, cyclists, trucks, and vans. It captures diverse traffic scenarios, making it ideal for autonomous driving applications.

3 Methodology

3.1 Dataset Exploration

To better understand the KITTI dataset used in our object detection pipeline, we analyzed four key aspects: **class distribution**, **bounding box sizes**, **image brightness**, and **image resolution**. While it is common to address data imbalance in machine learning, we intentionally chose **not to balance** the dataset for the reasons explained below.

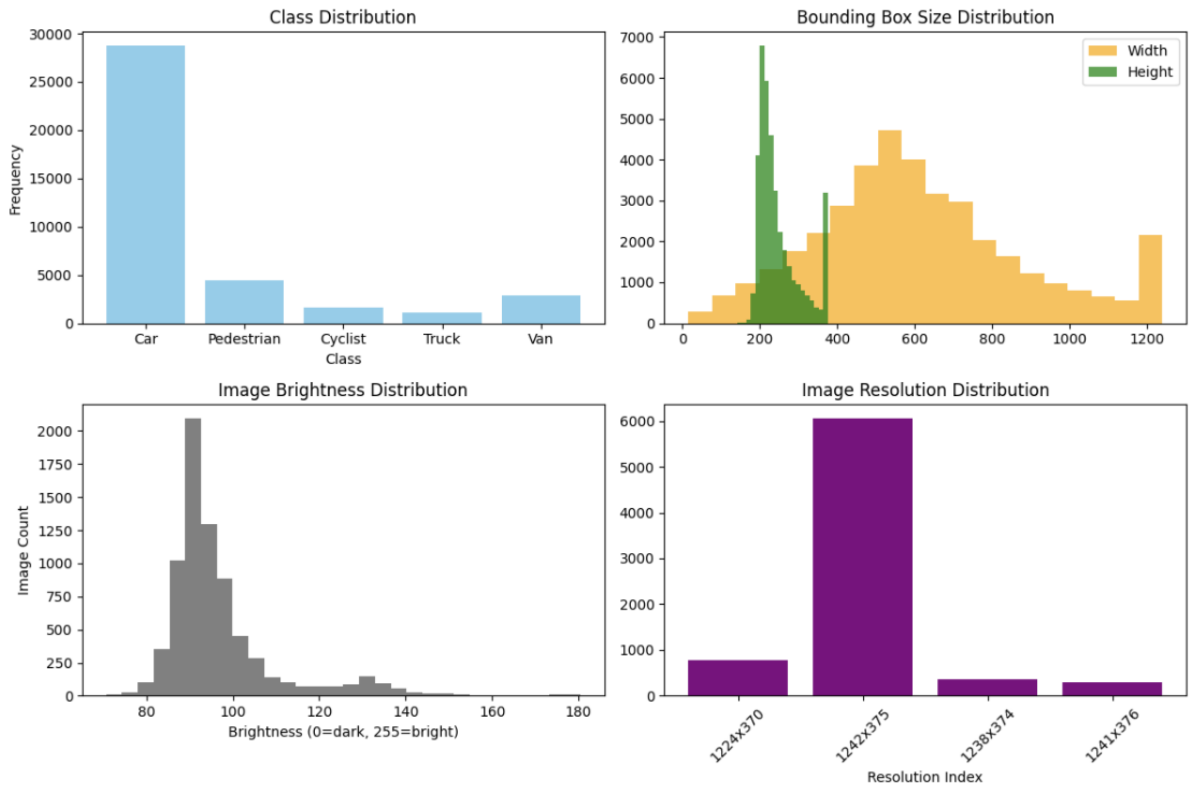

Figure 1: Dataset analysis: Class distribution, bounding box sizes, image brightness, and resolution.

3.1.1 Class Distribution

The dataset is heavily skewed toward the Car class, with fewer instances of Pedestrian, Cyclist, Truck, and Van.

Why we did not balance it:

- This imbalance reflects real-world traffic environments where cars are more frequent than other objects.
- Artificially balancing the data (e.g., by oversampling minority classes or under-sampling the majority class) could distort this natural distribution and result in unrealistic training scenarios.
- We relied on data augmentation and the robustness of the YOLOv11 architecture to ensure minority classes were still effectively learned.

3.1.2 Bounding Box Size Distribution

The histograms of bounding box widths and heights show a wide range of object sizes, which correspond to varying distances from the camera.

Why we did not balance it:

- Object size in the dataset is inherently tied to the real-world context (e.g., small objects often represent distant or smaller entities like pedestrians).

- Altering this distribution would risk degrading the model's ability to generalize to realistic object scales.

3.1.3 Image Brightness Distribution

Most images exhibit similar brightness levels (centered around 70–100), with a few brighter samples.

Why we did not balance it:

- Instead of balancing brightness, we applied data augmentation (e.g., brightness and contrast adjustments) to simulate diverse lighting conditions.

3.1.4 Image Resolution Distribution

The vast majority of images are sized at 1242×375, with minor variations present.

Why we did not balance it:

- All images were resized to a uniform input size (416×416) during preprocessing, making resolution balancing unnecessary.

3.2 Data Preprocessing and Augmentation

Data augmentation was applied to enhance model robustness, simulating varied driving conditions.
KITTI annotations were converted to YOLO format, normalizing bounding box coordinates. Images were resized to 416×416 and split into 80% training and 20% validation sets.

3.3 Why YOLO?

YOLO (You Only Look Once) was selected for its balance of speed and accuracy, crucial for real-time autonomous driving applications. Unlike two-stage detectors like Faster R-CNN, YOLO processes images in a single pass, achieving high frames per second (FPS) while maintaining competitive accuracy, making it ideal for resource-constrained environments.

3.4 Model Selection: YOLOv8, YOLOv10, and YOLOv11

We tested YOLOv8, YOLOv10, and YOLOv11. YOLOv8 provided a solid baseline for accuracy and speed. YOLOv10 improved efficiency but showed lower accuracy on KITTI. YOLOv11 excelled with the highest mean Average Precision (mAP), better detection of small objects like pedestrians, and a good balance of speed, making it the best choice for our project.

3.5 Model Training

The YOLOv11 model was trained for 120 epochs with a batch size of 16, using pre-trained COCO weights for transfer learning. Training metrics were logged with MLflow. YOLOv8, YOLOv10, and YOLOv11 were trained under similar conditions for comparison.

3.6 Model Export

The trained YOLOv11 model was exported to ***.pt*** format for deployment.

4 Results and Discussion

We evaluated YOLOv8, YOLOv10, and YOLOv11 on the KITTI dataset, focusing on recall, FPS, mAP50, box precision (IoU), and mAP50:95. The comparison is summarized in Table 1.

Model	**Recall**	**FPS**	**mAP50**	**Box Precision (IoU)**	**mAP50:95**
YOLOv8	0.7901	40.04	0.8632	0.8865	0.6244
YOLOv10	0.7532	44.87	0.8331	0.8875	0.5801
YOLOv11	0.8184	43.21	0.8975	0.9295	0.6751

Table 1: Comparison of YOLOv8, YOLOv10, and YOLOv11 on the KITTI dataset.

YOLOv11 outperformed the other models in most metrics, achieving the highest recall (0.8184), mAP50 (0.8975), box precision (0.9295), and mAP50:95 (0.6751).

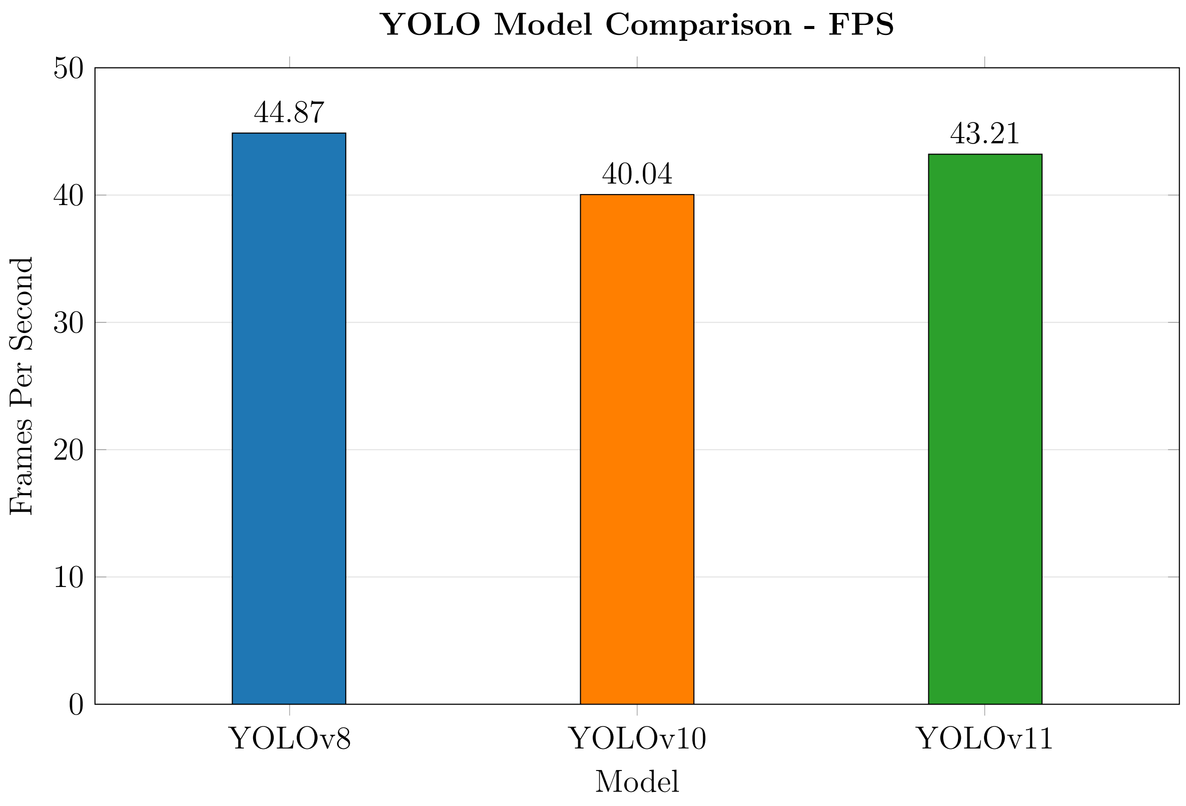

Figure 2: Comparison of YOLO models based on frames per second (FPS) performance.

While YOLOv10 had the highest FPS (44.87), YOLOv11's FPS (43.21) was still suitable for real-time applications, and its superior accuracy metrics justified its selection. YOLOv8 performed well but was outclassed by YOLOv11 in accuracy, despite a competitive FPS (40.04).

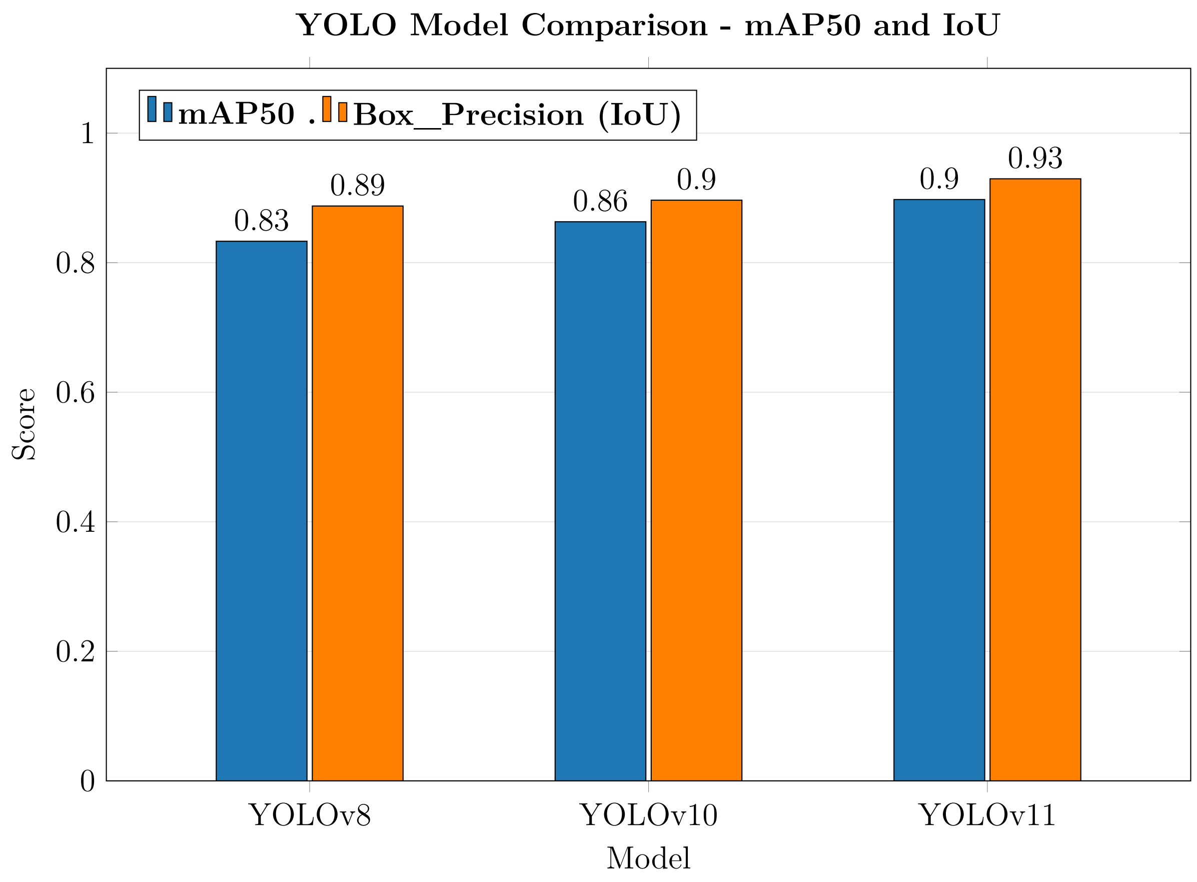

Figure 3: mAP50 and Box Precision (IoU) comparison of YOLOv8, YOLOv10, and YOLOv11. YOLOv11 leads with mAP50 (0.8975) and box precision (0.9295).

Figure 3 highlights YOLOv11's lead in mAP50 and box precision, underscoring its accuracy advantage.

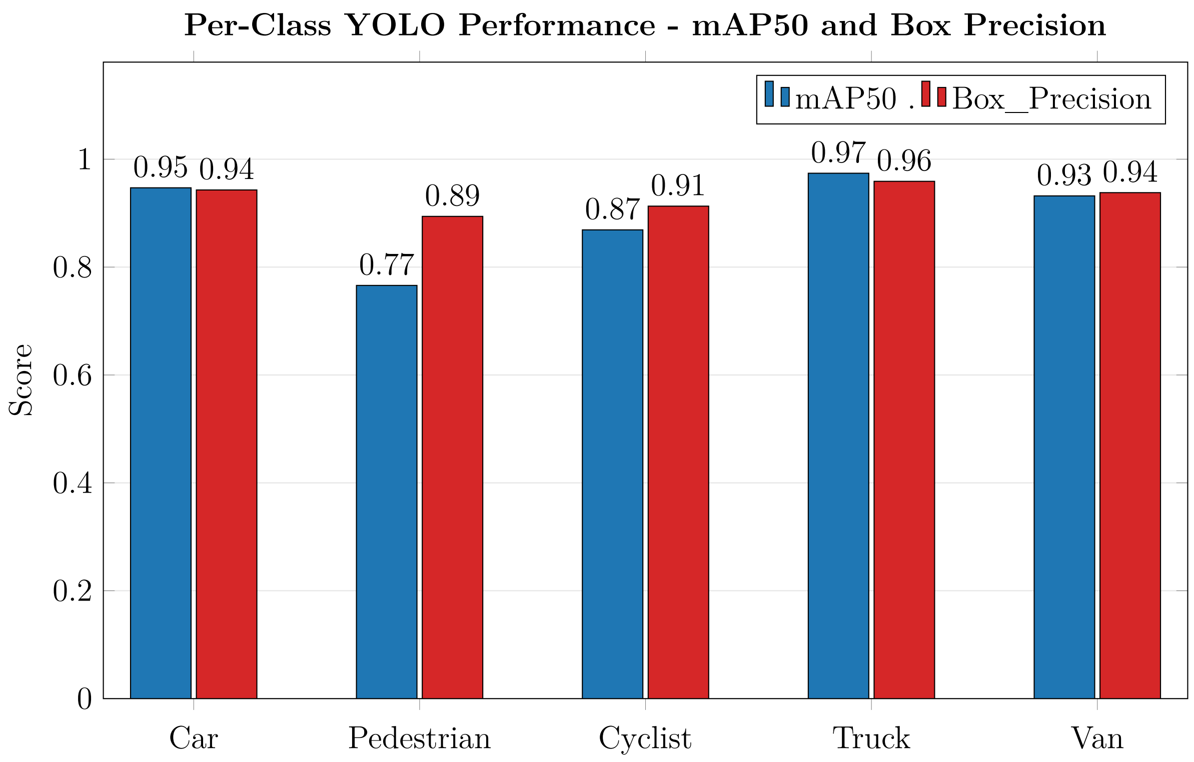

Figure 4: Per-class mAP50 and Box Precision for YOLOv11 across KITTI classes (Car, Pedestrian, Cyclist, Truck, Van). Performance is strong, with slight variations.

Figure 4 shows YOLOv11's per-class performance, with strong results across all classes, though small objects like pedestrians showed slight variations, suggesting potential for further optimization.

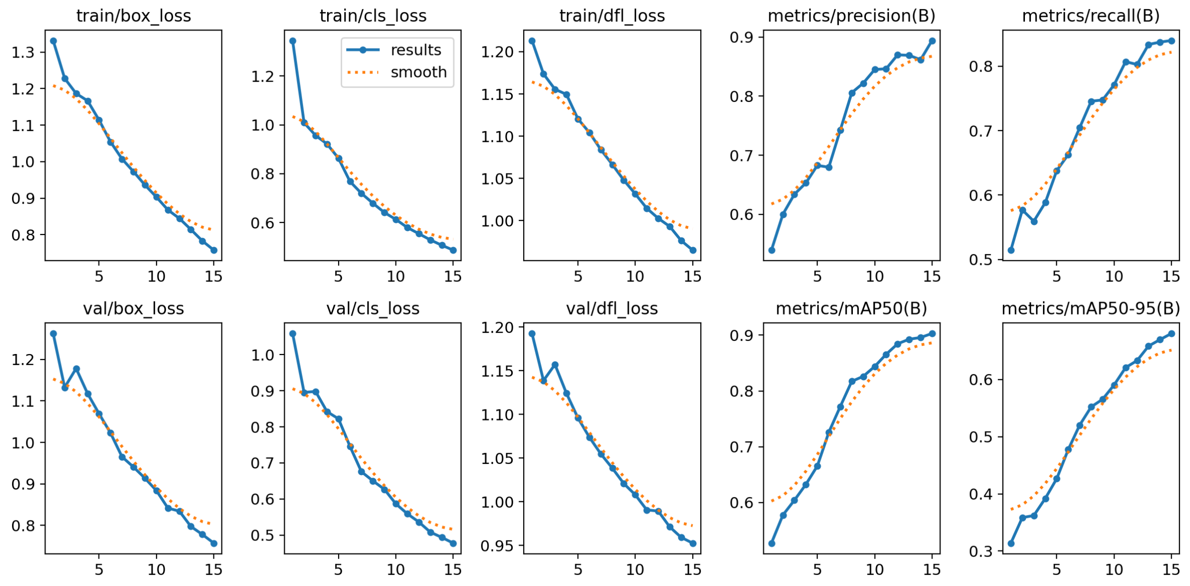

Figure 5: Training and Validation Metrics for YOLOv11 over 120 Epochs. Top row: train/box_loss, train/cls_loss, train/dfl_loss, metrics/precision(B), metrics/recall(B). Bottom row: val/box_loss, val/cls_loss, val/dfl_loss, metrics/mAP50(B), metrics/mAP50:95(B).

For YOLOv11, Figure 5 illustrates the training and validation metrics over 120 epochs. The top row shows training box loss, classification loss, distribution focal loss, precision, and recall, while the bottom row displays validation metrics and mAP scores. Losses decreased steadily, while precision, recall, and mAP increased, indicating effective learning. The model performed well on the KITTI dataset, but small objects like pedestrians posed challenges, suggesting potential for further optimization, such as additional augmentation or fine-tuning.

5 Conclusion

This project successfully developed a real-time object detection system using YOLOv11 on the KITTI dataset. YOLO was chosen for its speed and accuracy, with YOLOv11 selected for its superior performance across metrics. The model achieved high recall, mAP, and box precision, making it suitable for autonomous driving. Future work could focus on improving small object detection and testing on additional datasets.