Object Detection for Autonomous Driving: A YOLOv11n Approach

Fitsum Amine Nerayo
fitsum.nerayo@student.uni-luebeck.de

Bey Ishak
ishak.bey@student.uni-luebeck.de

Abstract

This report documents the development and deployment of a 2D object detection network to identify cars from images derived from a downscaled KITTI dataset. The project employs the YOLO (You Only Look Once) architecture, YOLOv11n, which is a state-of-the-art model renowned for striking a balance between speed and accuracy.

1. Introduction

Object detection is a fundamental task in computer vision with crucial roles in a wide variety of applications, including autonomous vehicles, robots, and surveillance security. In autonomous vehicles, object detection surrounding them, such as automobiles, pedestrians, and cyclists, in real-time and correct manner is important for secure travel and decision-making. This project focuses on the implementation of a 2D automobile object detector from camera data for a reduced, downscaled version of the KITTI dataset.

The KITTI dataset is a well-established benchmark for computer vision tasks in self-driving, offering various real-world situations. Our goal is to train an efficient object detection model capable of generalizing well to novel data. The detector's performance will be evaluated quantitatively using the Mean Average Precision (mAP) metric, which is a common evaluation metric in object detection competitions such as PASCAL VOC. To make the task simpler but still challenging, we are provided with pre-processed training and test data with test labels suppressed. This calls for a careful split of the provided training data into training and validation sets for internal performance estimation.

2. Related Work

The object detection community has seen some breakthroughs of late, driven mostly by deep learning. The conventional methods used to be a two-stage affair where a region proposal network (RPN) would detect potential object locations first and then a classification and bounding box regression step. Good examples are R-CNN, Fast R-CNN, and Faster R-CNN [1]. They are extremely accurate but also very computation-hungry with real-time applicability in autonomous driving being limited.

Single-shot detectors, however, predict bounding boxes and class probabilities directly from image features in a single pass, offering faster inference speeds. The YOLO (You Only Look Once) family of models revolutionized real-time object detection. YOLO has undergone numerous iterations since its inception, with each subsequent iteration outdoing the previous ones in either speed or accuracy, or in most cases, both. YOLOv1, the original model, framed object detection as a regression problem, predicting bounding boxes and class probabilities directly from whole images. Subsequent versions, YOLOv3, YOLOv4, and recent YOLOv5, YOLOv7, and YOLOv8, have introduced various architectural enhancements, including deeper backbones, improved activation functions, and more intricate loss functions, to drive real-time detection boundaries.[3]

For the sake of this project, we have selected YOLOv11n (nano edition of YOLOv8) from the ultralytics repository. This is motivated by the fact that it is a lightweight and performant model within the family of YOLO models and therefore ideal for quick experimentation and deployment, especially in resource-constrained environments such as Google Colab. The ultralytics implementation provides an out-of-the-box, highly optimized environment for training and inference, which aligns well with the project's requirements to adapt existing online code.

3. Method

Our object detection approach is based on training a YOLOv11n model[2]. The overall method consists of loading data, splitting data, dataset preparation in YOLO form, training the model, and finally inference on the test set.

3.1. Data Loading and Initial Inspection

The raw data exist as NumPy arrays: x_train.npy (training images), y_train.npy (test images), and x_{train}.npy (training labels). The files are read from a mounted Google Drive folder. The x_train.npy file contains dictionaries per image, with keys for 'boxes' (in the [ymin, xmin, ymax, xmax] format) and 'classes'. One of the most important preprocessing steps, already completed, was the removal of labels for objects that are not ”cars.”

3.2. Evaluation Metric: Mean Average Precision (mAP)

In order to quantify the detector’s performance, we utilize the Mean Average Precision (mAP) metric, a standard for object detection. The function calculates:

- **Intersection over Union (IoU):** The bbox_iou function computes the overlap between predicted and ground truth bounding boxes. A prediction is considered correct if its IoU with a ground truth box exceeds a threshold (typically 0.5).
- **Precision-Recall Curve:** For each class, precision and recall values are computed across various confidence thresholds.
- **Average Precision (AP):** The area under the precision-recall curve
- **Mean Average Precision (mAP):** The average of the APs across all detected classes.
- **YOLOv11n Model and Implementation:** We use the ultralytics library for YOLOv11n to make the training and inference pipeline extremely easy.

4. Evaluation

4.1. Experimental Setup

Training data was split into training and validation sets to monitor generalization ability and prevent overfitting. The images were then saved as JPEG files. A custom function, was implemented to transform the original bounding box format into the YOLO format. This function includes robust error handling for invalid box coordinates (e.g., zero area or invalid aspect ratio) by clipping coordinates and enforcing minimum sizes, thereby ensuring data quality for training. After training, the network performance was evaluated through computation of the Mean Average Precision (mAP) score. The model that was trained was then used to predict on a held-out test set, with the specified output format being used for bounding boxes, classes, and confidence scores. The processed labels were then saved as .txt files. Figure 1 shows sample training images with their ground truth labels, demonstrating the input to the training process.

4.2. Discussion

The training process used 100 epochs, where the key measures were recorded in csv file (Figure 2). The box and class training and validation losses always reduced, which suggests that the model was learning adequately to identify cars. Specifically, validation box loss and class loss had a linear decrease, which suggests excellent generalization without significant overfitting by the by the end of 100 epochs.

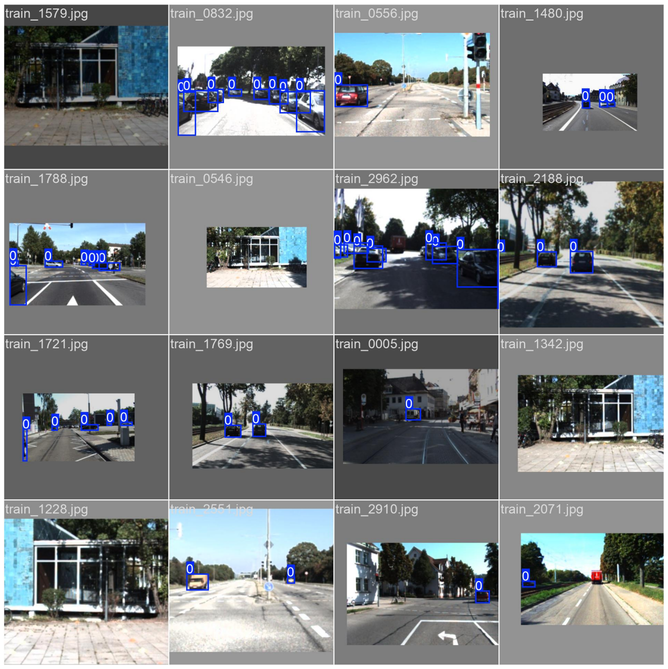
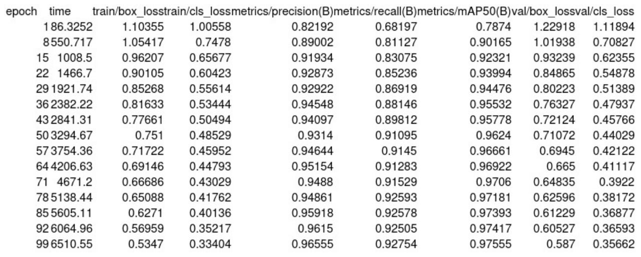


Figure 1. Sample batch of training images from the KITTI dataset showing ground truth bounding boxes and class labels



epoch	time	train/box_loss	train/cls_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	val/box_loss	val/cls_loss
1	86.3252	1.10355	1.00558	0.82192	0.68197	0.7874	1.22918	1.11894
8	550.717	1.05417	0.7478	0.89002	0.81127	0.90165	1.01938	0.70827
15	1008.5	0.96207	0.65677	0.91934	0.83075	0.92321	0.93239	0.62355
22	1466.7	0.90105	0.60423	0.92873	0.85236	0.93994	0.84865	0.54878
29	1921.74	0.85268	0.55614	0.92922	0.86919	0.94476	0.80223	0.51389
36	2382.22	0.81633	0.53444	0.94548	0.88146	0.95532	0.76327	0.47937
43	2841.31	0.77661	0.50494	0.94097	0.89812	0.95778	0.72124	0.45766
50	3294.67	0.751	0.48529	0.9314	0.91095	0.9624	0.71072	0.44029
57	3754.36	0.71722	0.45952	0.94644	0.9145	0.96661	0.6945	0.42122
64	4206.63	0.69146	0.44793	0.95154	0.91283	0.96922	0.665	0.41117
71	4671.2	0.66686	0.43029	0.9488	0.91529	0.9706	0.64835	0.3922
78	5138.44	0.65088	0.41762	0.94861	0.92593	0.97181	0.62596	0.38172
85	5605.11	0.6271	0.40136	0.95918	0.92578	0.97393	0.61229	0.36877
92	6064.96	0.56959	0.35217	0.9615	0.92505	0.97417	0.60527	0.36593
99	6510.55	0.5347	0.33404	0.96555	0.92754	0.97555	0.587	0.35662

Figure 2. Training progress metrics showing losses, precision, recall, and mAP

The quantitative performance of the trained model is also demonstrated in the Precision-Recall curve (Figure 3). The model achieved a strong mAP@0.5 of 0.975 for the ’car’ class (as ’car’ is the only class), indicating a high degree of accuracy at an Intersection over Union (IoU) threshold of 0.5. This high mAP suggests that the model is very effective at correctly identifying cars and accurately localizing them.

The Confusion Matrix (Figure 4) provides a detailed breakdown of the model’s classification performance.

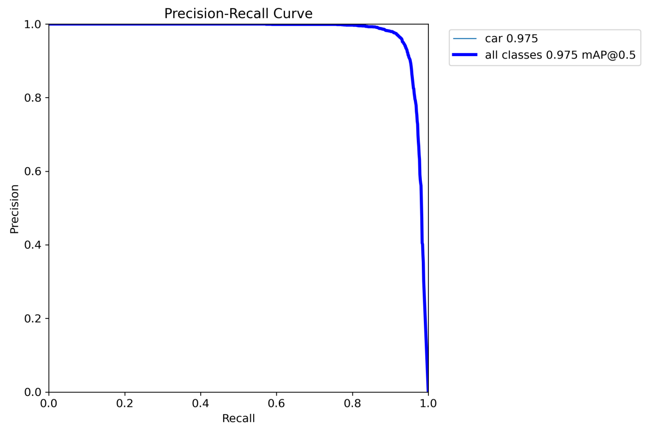


Figure 3. Precision-Recall (PR) curve for the ’car’ class (mAP@0.5 = 0.975).

- **True Positives (car - car):** 2411 instances, representing correctly detected cars.
- **False Negatives (background - car):** 127 instances, where the model failed to detect a car. These could be due to challenging conditions (e.g., heavy occlusion, small objects, poor lighting).
- **False Positives (car - background):**: 244 instances, where the model incorrectly identified a background region as a car. These might include objects that look similar to cars or errors in complex scenes.

The relatively low number of false negatives and false positives, especially compared to the high number of true positives, corroborates the high mAP score and indicates a robust detector. While the model performed exceptionally well, false positives often arise from objects with car-like features (e.g., parts of buildings, signs), and false negatives can be attributed to heavily occluded vehicles or objects at extreme distances, which are inherently difficult to detect even by the human eye.

4.2.1 Inference on Test Set:

After training, the model saved its best weight. The trained model was loaded to perform inference on the test set. The model.predict function was used with a confidence threshold (conf=0.5) during training. The model predictions (the bounding boxes, class IDs, and confidence scores) for each test image were obtained. Figure 5 shows an example of the model prediction on a test image, clearly showing the ability of the model to detect and locate cars using high confidence scores.

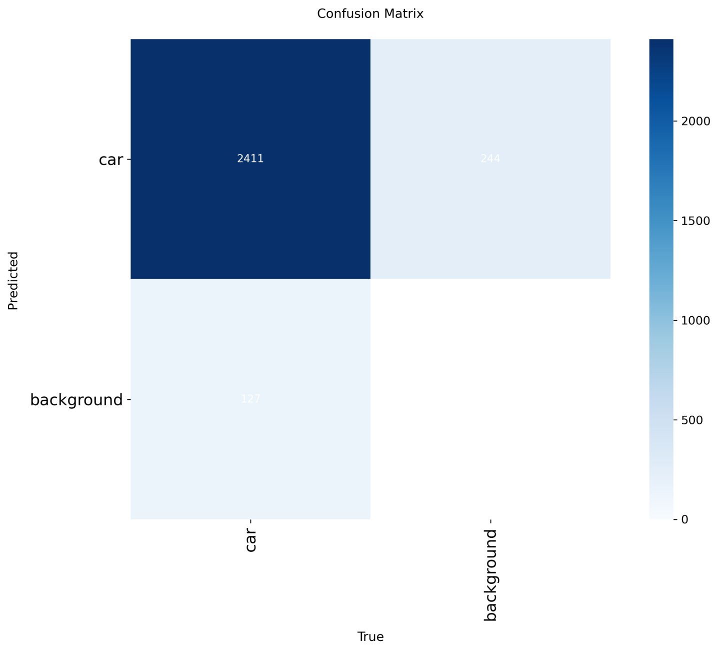


Figure 4. confusion matrix for the ’car’ class.

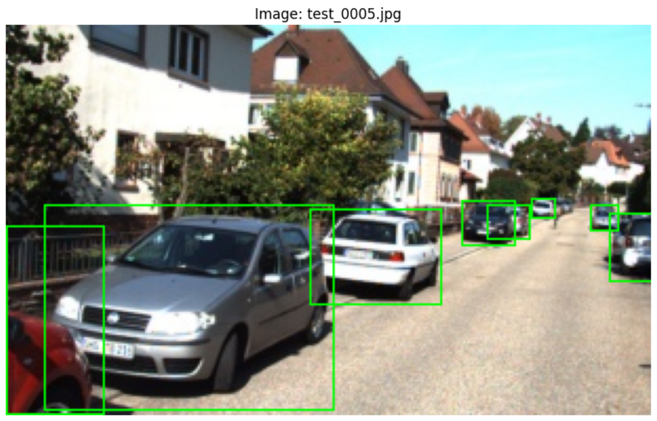


Figure 5. Sample test image with predicted bounding boxes

5. Conclusion and Future Work

This project was able to train and deploy a 2D object detector using YOLOv11n for the detection of cars on a downscaled KITTI dataset. Through careful data preparation, including data splitting into training and validation sets, as well as formatting annotations into YOLO, we were able to train a robust model. The use of the ultralytics framework significantly eased the development process, where we were able to train as well as use it for inference efficiently. The model achieved a high mAP@0.5 of 0.975 in the validation set, indicating its strong performance. The predicted bounding boxes, classes, and confidence scores were successfully generated in the required format for the held-out test set.

For future work, several directions could be explored to continue to enhance the performance of the detector:

- **Larger Model Variants:** If computational resources permit, experimenting with larger YOLOv11 variants (e.g., YOLOv11s, YOLOv11m) or exploring knowledge distillation could potentially lead to even higher accuracy at the cost of increased inference time, while still aiming for real-time performance.
- **Multi-Class Detection:** Extending the detector to identify other critical classes like pedestrians, cyclists, and traffic signs would be a natural next step for a comprehensive autonomous driving perception system.
- **Error Analysis and Mitigation:** A more detailed scrutiny of the false negatives and false positives, preferably by categorizing the type of challenging cases, could potentially be utilized to guide targeted data augmentation or model improvements.

References

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, and Ezgi Mercan. R-cnn for object detection. In *IEEE Conference*, 2014.

[2] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.