Jason Bone
CECS 385
Dr. Aysegul Cuhadar
30 November 2025

Analysis of YOLOv8 vs. Classic Object Detectors on KITTI Autonomous Driving Dataset

Abstract:

This study uses the KITTI autonomous driving benchmark to compare the performance of YOLOv8 against classic object detection architectures such as Faster R-CNN, SSD and other versions of the YOLO model on the KITTI autonomous driving benchmark. Our YOLOv8 model achieved 90.2% mAP@0.5 on the KITTI validation set, which shows significant improvements from other models such as Faster R-CNN at 78.1%, SSD at 75.3% and YOLOv3 at 72.1% reported. This research highlights the evolution of real-time object detection that is critical for applications in autonomous driving.

Introduction and Problem Statement:

For autonomous driving systems to be trusted, there must be robust real-time object detection to handle to complex environments that happen on the roads. The KITTI dataset [1] is the standard benchmark for evaluating automotive perception systems. There are many object detection architectures out there that are constantly being worked on and improved; however, their comparative performance on specific automotive tasks remains unclear. Our research aims to answer: How does the modern YOLOv8 architecture compare to other established detectors on the KITTI autonomous driving benchmark?

Research Questions:

1. How does the performance of YOLOv8 compare to other models on KITTI?
2. How have these object detectors evolved to be more well-suited for real-time applications?
3. What are the practical implications for autonomous driving systems?

Literature Review:

Datasets:

The KITTI Vision Benchmark Suite [1] provides a standardized evaluation for autonomous driving models. The KITTI dataset contains 7,481 training and 7,518 testing images, all collected from real driving scenarios with annotations for types of vehicles, pedestrians and cyclists. There are complementary datasets such as Common Objects in Context (COCO) [5], which contains 330,000 images across 80 categories in everyday context. Datasets like COCO can offer a pre-training foundation for these self-driving models and offer more general benchmarks.

Figure 1: A sample image from the COCO Dataset.

Figure 2: A Sample image from the KITTI Dataset.

Models:

The Faster Region-based Convolutional Neural Network (Faster R-CNN) [2] was revolutionary in object detection because it introduced the Region Proposal Network (RPN). Faster R-CNN's predecessors had to use external region proposal methods such as Selective Search. Faster R-CNN works in two stages. The first is that the RPN generates candidate regions in which objects are likely to be then a separate network will classify the objects and refine the region selected. While Faster R-CNN was able to achieve great accuracy, significant computational

overhead was introduced due to the sequential nature of its pipeline. The RPN and the main detection network share processing work, allowing the entire system to be trained all together while still achieving very high accuracy. The two-step nature of the Faster R-CNN allows for these great accuracy scores, but also creates a bottleneck that limits the model's ability for real-time use, achieving only 5-7 frames per second when real-time applications require at least 30 frames per second.

Single Shot Multibox Detector (SSD) [3] is an earlier single-stage detection model. These single-stage models, such as SSD, marked a shift by performing detection in one single pass rather than using the region proposal stage. These models use multiple feature maps of different sizes of the image to detect objects of all sizes. Each location has predefined boxes in which class probabilities and predicted and bounding boxes (bbox) are refined. This design has a much more favorable speed-accuracy trade-off off being able to achieve 46 frames per second with competitive accuracy scores. Even with the differently scaled featured maps, SSD still struggled to detect smaller objects.

You Only Look Once (YOLO) version 3 [4] improved upon the single-stage model by introducing a more sophisticated backbone in which residual connections allow more layers of feature maps. The more feature maps a model is able to use in training, the more likely its classifications will be correct; however, the bounding boxes for these objects become much more fuzzy with more layers due to the different sizes of these maps. Residual connections allow the more accurate spatial data from early maps to be combined with the highly accurate classifications from later maps with much less loss. This version splits the image into a grid and then predicts classes and object edges at around 35 frames per second, and accuracy scores competitive with two-stage models.

YOLOv4 [6] improved on the single-stage model by testing many different improvement techniques and logging them all into two main categories. One is a "bag of freebies," meaning any strategy they used that boosts accuracy without affecting speed, such as advanced data augmentation and optimized loss functions. The other is a "bag of specials" meaning any modification that delivers significant gains in accuracy while having minimal computation loss. The methodical approach to improving the model allowed state-of-the-art accuracy alongside the real-time speed needed for practical applications.

YOLOv8 [7] is one of the latest versions of the YOLO model. This version predicts bounding boxes based on object centers and their size rather than

predicting offsets from predefined anchor boxes. This shift makes the model much less sensitive to hyperparameters and improves training stability. This model improves on the backbone, allowing for maintained special integrity while improving classification. The redesigned detection allows the model to both classify and fit the box at the same time, leading to truly more accurate predictions rather than a lucky guess. Beyond the technical improvements, training for the YOLOv8 model has been made much more developer-friendly. Faster training times, state-of-the-art accuracy, and ease of use for developers have all contributed to YOLOv8 being a very popular choice in both research and industrial applications.

Methodology:

Experimental Design:

We are comparing our implementation of a YOLOv8 model pretrained on a 15 thousand image subset of the COCO dataset versus the reported Faster R-CNN [2], SSD [3], and YOLOv3 [4]. The reported statistics from the model come from the official KITTI test set, while our YOLOv8 model used an 80/20 split from the training set due to the private nature of the true KITTI test set. This limitation should be considered when interpreting comparative performance. We are comparing these models based on four key metrics being mean average precision at intersection over union equal or greater than 50% (mAP@0.5), mean average precision at intersection over union from 50% to 95% (mAP@0.5:0.95), precision, and recall. mAP@0.5 is how accurate the model is when the predicted box overlaps the true box by at least 50%. This is the benchmark metric for object detection as it balances finding all objects with the detection of real objects. A score of 90% would mean that nine out of ten objects are detected correctly with good placement of the predicted box. mAP@0.5:0.95 averages the accuracy across ten different overlap thresholds. This metric measures truly how accurate your model is, not by classification but by placement. Precision is the percent of correct detections, while recall is the percent of total objects that are detected.

Implementation Details:

First step to train our YOLOv8 model was ensuring that all annotations for both COCO and KITTI datasets match the format of the YOLO model (<class> <x_center> <y_center> <object width> <object height>, all normalized between 0 and 1). COCO's annotations originally came in a single JSON containing images: [image_id, file_name, image height, image width], annotations: [image_id, bbox=[x_min, y_min, object width, object height], segmentation, …], absolute pixel coordinates, and many other details. Mapped the 80 COCO categories to YOLO's class index format and eventually filtered to the three classes consistent with KITTI. Due to the difference in the number of classes, the actual true weights from pretraining were discarded, leading to diminished returns

from pretraining, but the general visual features learned were kept, still providing some value. Created a corresponding .txt image that contains one line per bounding box matching the YOLO format. Then finally generated the file structure expected by the model.

```
coco/
├── images/
│   ├── train2017Subset/
│   └── val2017/
└── labels/
    ├── train2017Subset/
    └── val2017/
```

KITTI already came in .txt format; however had many details that YOLO did not care for, so we had to extract only the necessary info from each annotation. Our script mapped the classes, “Car” to zero for YOLO format, extracted columns four through seven as those are the bounding box values provided, and then put that information into a YOLO formatted txt. Then again, ensure the file structure matches what the model needs for training. Then the model pretrains on the COCO dataset. The model trains for 30 epochs on a 15 thousand image subset of the COCO dataset to help generalize the model. For the main training of the model, we use an 80/20 training/validation split of the KITTI dataset for 60 epochs in batches of 16.

Other Key Hyperparameters:

- Initial learning rate: 0.002
- Final learning rate: 0.01
- Momentum: 0.937
- Weight decay: 0.0005
- Warmup epochs: 3.0
- Image size: 768x768
- Optimizer: Auto
- Loss components: Box loss weight (7.5), class loss weight (0.5), DFL loss weight (1.5)
- Data augmentation: HSV adjustments (h=0.015, s=0.7, v=0.4), translation (0.1), scale (0.5), flip left-right (0.5), mosaic (1.0)

Everything was run using Ultralytics in a Google Colab environment with high memory. After training, YOLO automatically generates evaluation plots such as precision-confidence curves, precision-recall loss, F1-confidence curves, recall-confidence curves, confusion matrices, instance distribution histograms, bounding box size distributions and training loss curves. It also returns example detection images showing bounding boxes with their respective class labels and confidence scores.

Unfinished Faster-RCNN:

Originally plan was to also build a Faster R-CNN model to compare on our own data; however, when trying to run on Google Colab ran into issues that were not present on my local machine. Local resources were not powerful enough to train a model, and time constrains prohibited solving the issue within Colab so moved forward with comparing to reported models.

Comparative Analysis:

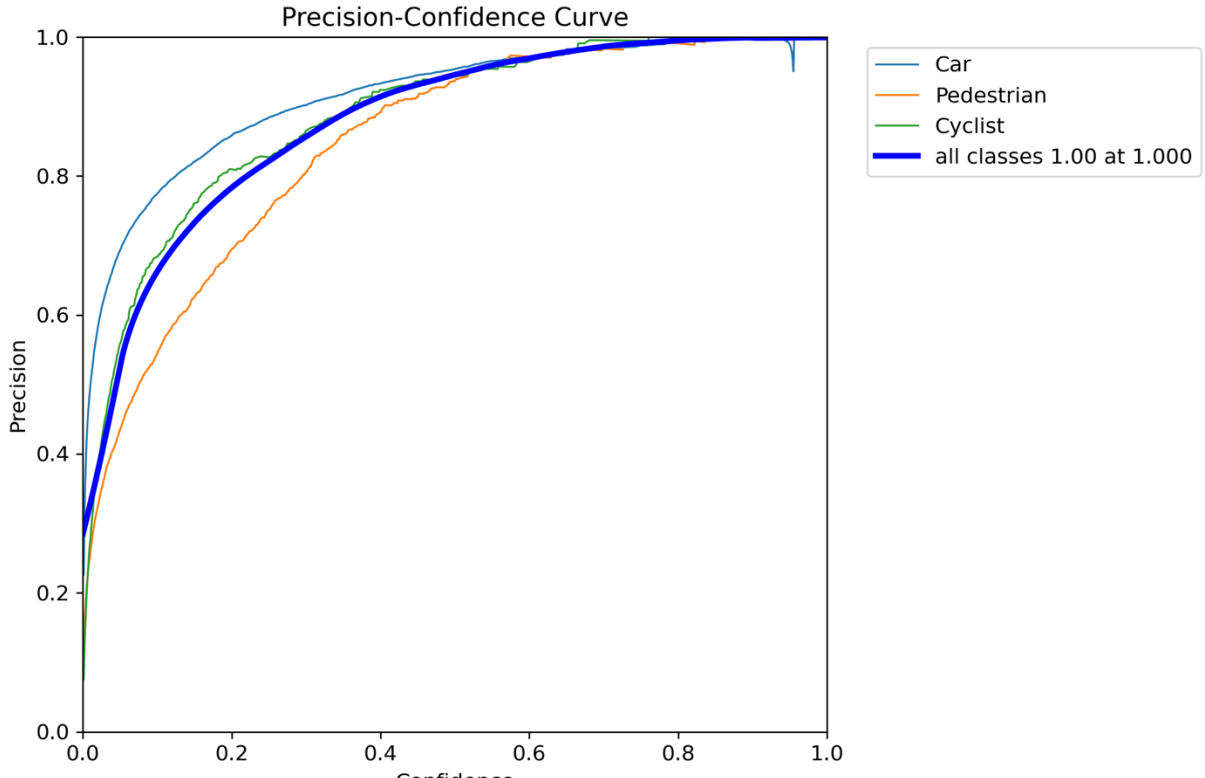
We are directly comparing our YOLOv8 model, trained and tested on an 80/20 train/validation split, to models tested on the official KITTI test set. To reduce discrepancies in evaluation protocols, analysis adheres to KITTI's official metrics.

Results and Analysis:

Model	mAP@0.5	mAP@0.5:0.95	Dataset	Source
YOLOv8 (Ours)	90.0%	64.3%	KITTI-val	This work
Faster R-CNN	78.1%	57.6%	KITTI-test	[2]
SSD	75.3%	54.2%	KITTI-test	[3]
YOLOv3	72.1%	51.8%	KITTI-test	[4]

Table 1: Key metrics comparison for reported KITTI results from community benchmarks and leaderboard submissions. https://www.cvlibs.net/datasets/kitti/

Results from the final epoch of the YOLOv8 model were precision reached 91.0%, recall reached 82.0%, mAP@0.5 90.0%, and mAP@0.5:0.95 64.3%. Training showed steady improvement across all epochs, with train box loss decreasing from 1.318 to 0.678 and classification loss from 1.467 to 0.408. Validation followed the same trend.



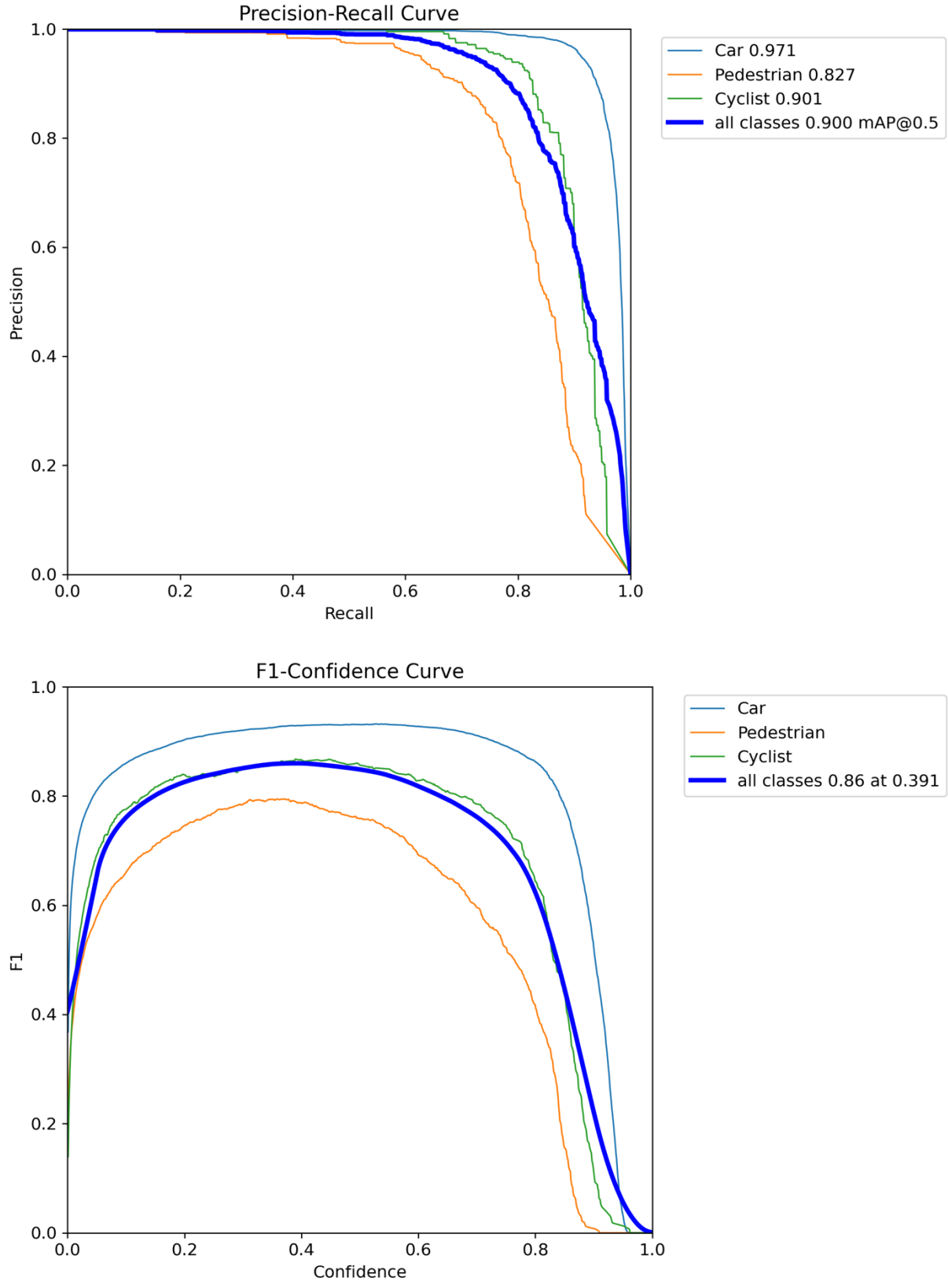


Figure 4: F1-Confidence Curve for the YOLOv8 model

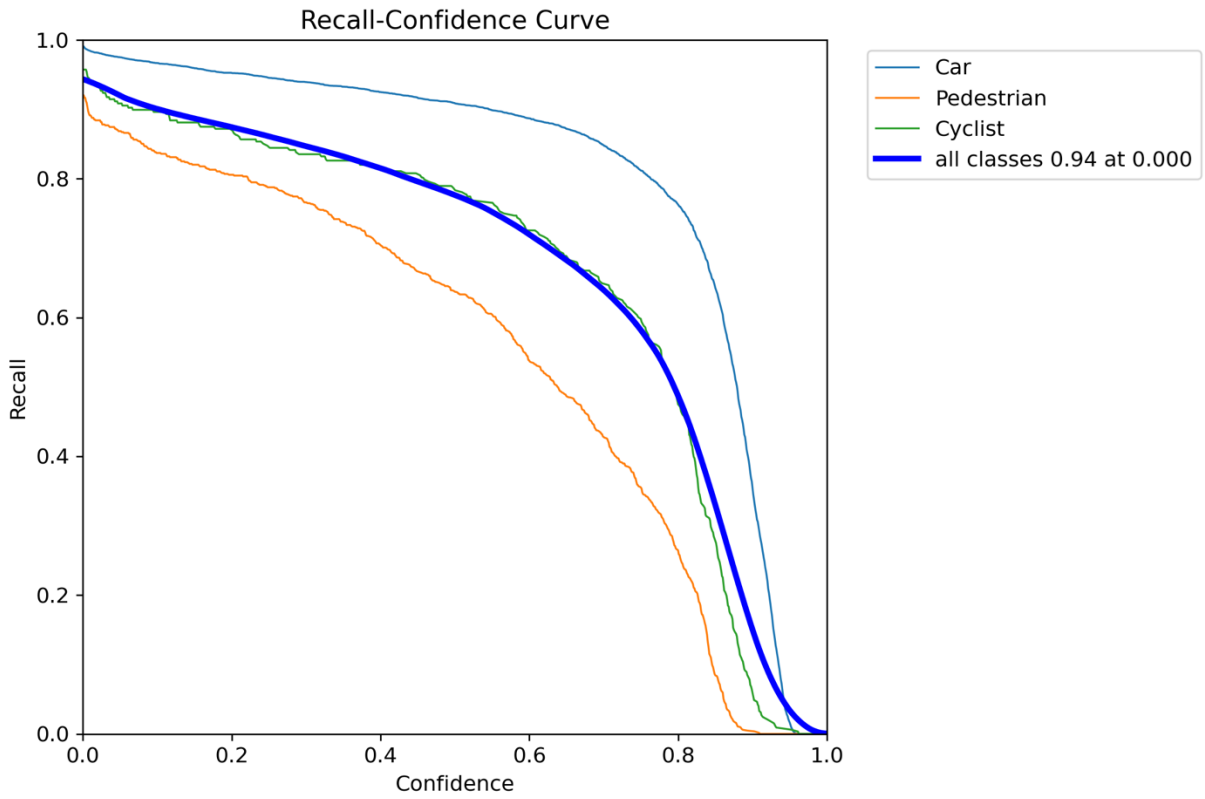


Figure 5: Recall Confidence Curve for the YOLOv8 model

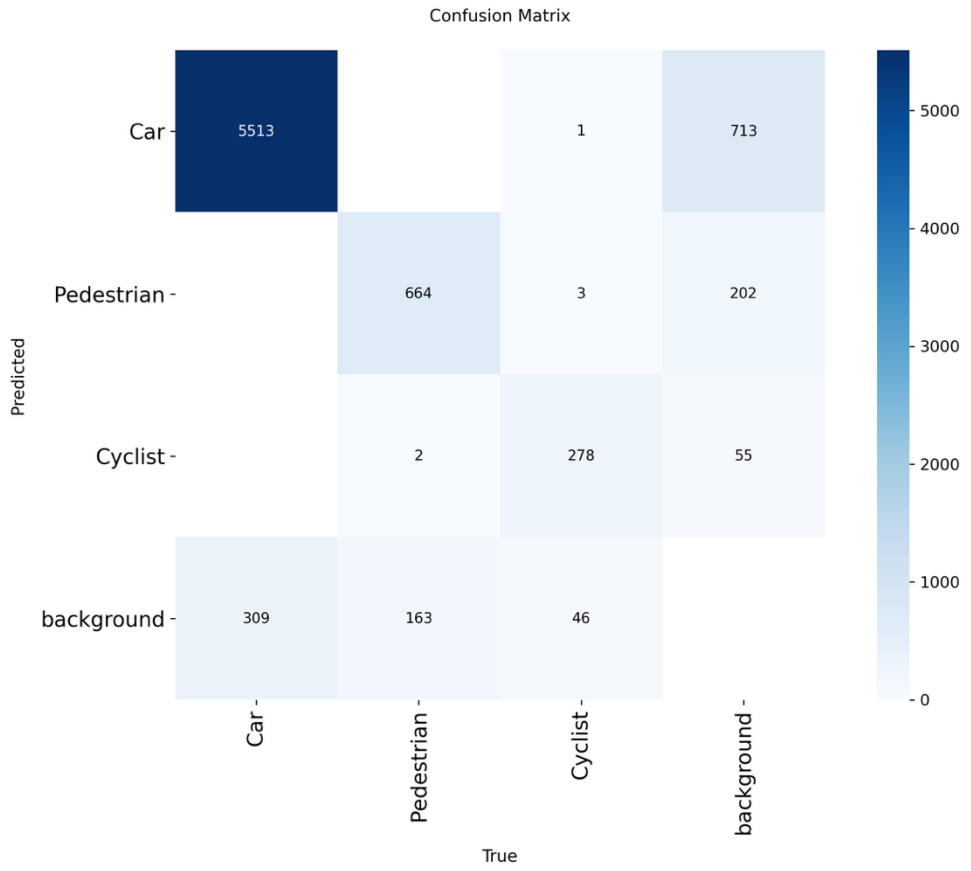


Figure 6: Confusion Matrix for the YOLOv8 Model

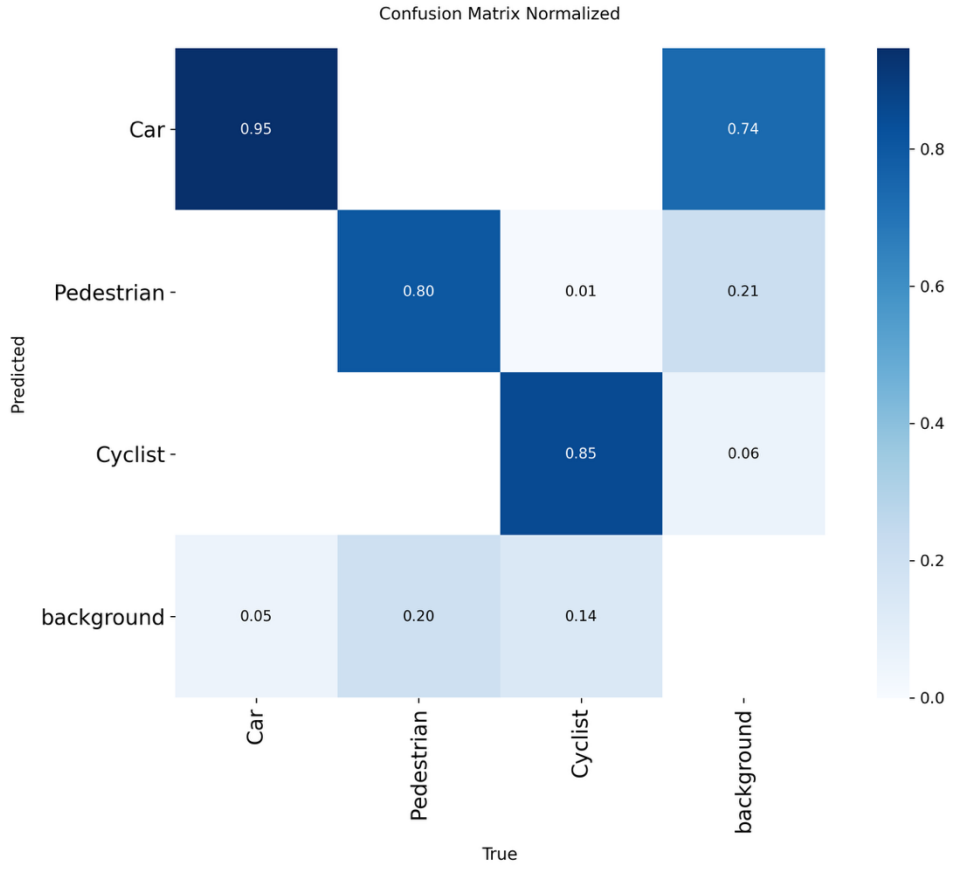


Figure 7: Normalized Confusion Matrix for the YOLOv8 Model

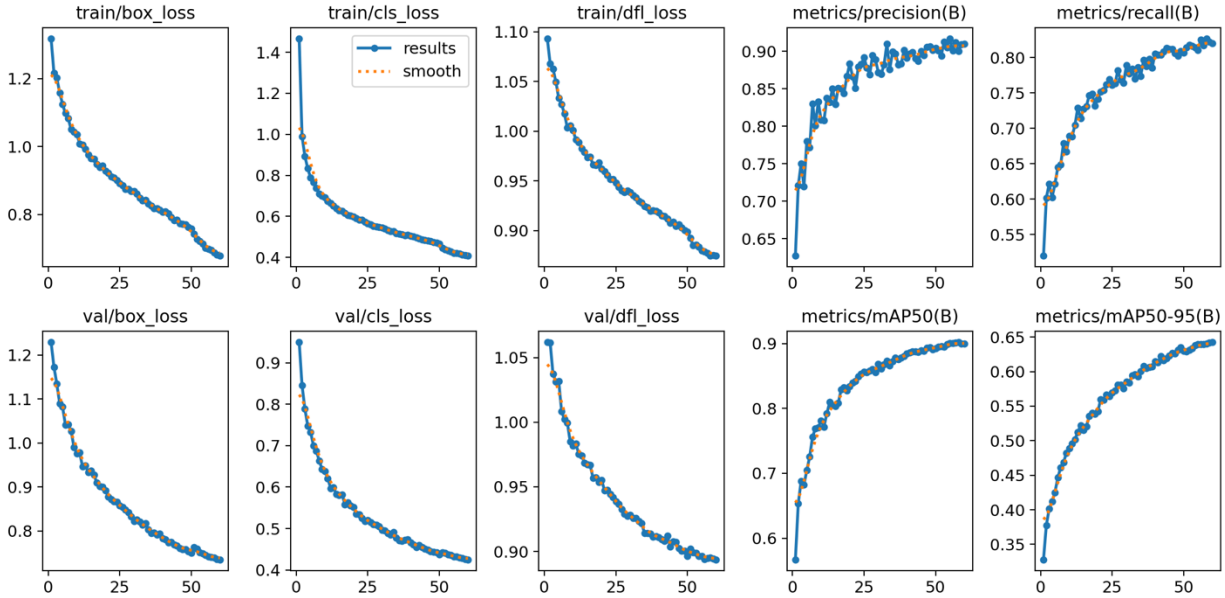


Figure 8: Loss Curves for the YOLOv8 model

Figure 9: Example detection images of the YOLO model

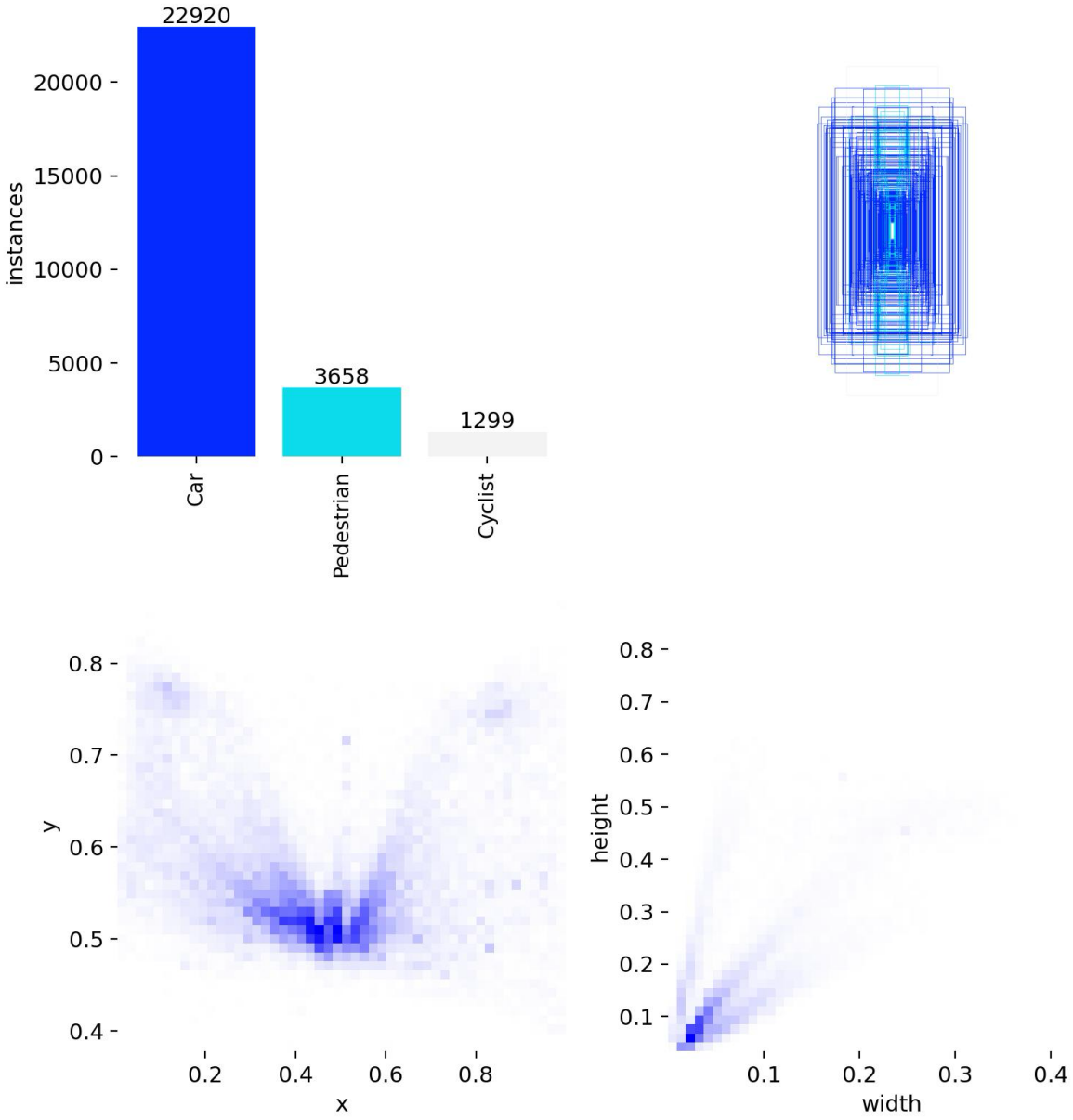


Figure 10: Label distribution for the YOLOv8 Model

Key Findings:

- YOLOv8 model outperforms the Faster R-CNN by around 12% in mAP@0.5 and 6.7% in mAP@0.5:0.95, SSD by 14.7% and 10.1%, and YOLOv3 by 17.9% and 12.5%, respectively. The model performed strongest on cars, followed by cyclists, then pedestrians.
- Precision-confidence curve (Figure 3) indicates reliable high confidence detections through high precision at confidence thresholds up to 0.6. The F1-confidence curve (Figure 4) peaks at 0.86 F1-score at a 0.391 confidence

threshold, which is optimal for balanced precision-recall in autonomous driving. Recall-confidence curve (Figure 5) illustrates very high recall at low confidence thresholds.

- The raw confusion matrix (Figure 6) shows the model had 5,513 true positives for cars, 664 for pedestrians, and 278 for cyclists. The normalized values (Figure 7) help showcase how this strong performance achieves 95% accuracy for cars, 80% for pedestrians, and 85% for cyclists. The most common confusions were pedestrians classified as background at 21% likely due to small object sizes.
- The loss curves (Figure 8) returned by YOLO show consistently improving results. Instance distribution confirms that cars are dominating with 22,920 instances (Figure 10).
- Detection examples (Figure 9) demonstrate how robust YOLOv8 is with accurately bounded objects at high confidence in clear conditions, though there are some low-confidence detections usually due to distant scenarios or clutter.

These findings help demonstrate how well YOLOv8 can handle KITTI's challenges.

Discussion:

YOLOv8's anchor-free design, along with the refined loss functions, enables much better generalization compared to classic image detection models. Also, the evolution of the YOLO model is showcased through YOLOv8's single-stage efficiency for real-time tasks. While the YOLO model's 90% mAP@0.5 is an impressive score, there is still a 10% error rate, which is not acceptable for autonomous driving purposes. YOLO models in general, but especially YOLOv8, are able to train at a much higher rate compared to two-stage models, highlighting their practicality for real-time detection. KITTI's focus on urban European roads limits what road conditions were seen by the model; this is still far too general for a real-world autonomous driving object detector.

Conclusion and Recommendations:

YOLOv8 shows significant improvement not only from classic models like Faster R-CNN or SSD, but also significantly improves on the former version of itself, achieving 90% mAP@0.5 while still having the real-time capabilities needed for autonomous driving.

Exploring how well YOLO could perform when trained on a high-powered machine without the need for Google Colab and the ability to use much larger datasets would benefit the autonomous driving industry. Continuing to optimize these single-stage models is the key to fully autonomous vehicles being fast and trustworthy.

Limitations:

Results from the YOLOv8 model are limited due to being a split from the training set rather than the true KITTI test set, and computational resources prohibited much of the training that could be done.

Reflection:

Throughout this research, not only have skills been built around these image detection models, including how to prepare data, train and evaluate these models, but also invaluable knowledge has been gained surrounding computational limits of my machine and the resources available, like Google Colab, to work around those limitations. I have grown as a researcher through new skills of experimental design and empirical evaluation, as well as furthering my understanding of how to navigate academic literature and synthesize findings.

References:

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in Proc. CVPR, 2012, pp. 3354–3361.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2017.

[3] W. Liu et al., "SSD: Single shot multibox detector," in Proc. ECCV, 2016, pp. 21–37.

[4] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv:1804.02767, 2018.

[5] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Proc. ECCV, 2014, pp. 740–755.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," arXiv:2004.10934, 2020.

[7] Ultralytics, "YOLOv8 documentation," 2024. [Online]. Available: https://docs.ultralytics.com