RGB Image Detection Model using YOLOv8 for Traffic Targets Detection in Autonomous Vehicles

Abhishek Kumar Gupta, Shruti Sangam, M. Brindha

National Institute of Technology, Trichy, India

Corresponding author: Abhishek Kumar Gupta, Email: abhishekofficial9576@gmail.com

This study uses YOLOv8 to propose an improved traffic target detection model for autonomous cars. Complex problems including identifying cars, pedestrians, and traffic signs in a variety of environments—including dim lighting and occlusions—are handled by the RGB image detection model. By utilizing the KITTI dataset and sophisticated data augmentation methods such as random erasure and normalizing, the model significantly increases the robustness of detection. With a mean average precision (mAP) of 0.84 and a high inference speed, YOLOv8 outperforms baseline models and is appropriate for real-time applications. In order to further increase detection accuracy in harsh environments, future research will concentrate on integrating sensor modalities like LiDAR. These findings demonstrate how YOLOv8 might improve autonomous systems' perceptive abilities, resulting in safer navigation.

Keywords: Traffic Target Detection, YOLOv8, Autonomous Vehicles, KITTI Dataset, Image Processing, Computer Vision.

1 Introduction

An important development in contemporary transportation, autonomous cars have the potential to completely transform the sector by providing safer, more effective, and more accessible travel options. The ability of autonomous systems to consistently sense and comprehend their surroundings is a crucial component. Accurate traffic target identification, which entails recognizing and categorizing a variety of objects in real time, including cars, pedestrians, and traffic signals, is a major component of this capability. In real-world driving situations, traditional computer vision techniques encounter numerous difficulties due to different lighting conditions, occlusions, and object appearances. Convolutional neural networks (CNNs), a type of deep learning model, have made considerable strides in the discipline by enhancing object detection in such dynamic environments. Because of its effectiveness and precision in real-time detection, the YOLO (You Only Look Once) series stands out among deep learning models. The most recent version of the YOLO series, YOLOv8, is perfect for traffic target detection since it includes advanced augmentation techniques, optimized training processes, and a number of architectural enhancements. This study makes use of YOLOv8's advantages to create a model that can effectively identify several traffic targets in challenging settings. The model is trained to correctly identify traffic objects in a variety of environments using the KITTI dataset, a well-known benchmark in autonomous driving research.

Computer vision applications, particularly autonomous driving, have advanced significantly as a result of the development of object detection models, especially YOLO (You Only Look Once). By offering a crucial dataset for performance testing, Geiger et al. (2012) presented the KITTI Vision Benchmark Suite, which lay the foundation for assessing autonomous driving algorithms. This dataset served as a foundation for creating and comparing models under a variety of driving conditions. The true innovation was YOLO, which was first shown by Redmon et al. (2016)[2] and completely changed real-time object identification. YOLO is a well-liked option for applications requiring real-time processing, such autonomous car navigation, because of its unified detection pipeline, which greatly increased speed and accuracy. The success of YOLO has been built upon by later iterations that have improved the ratio of accuracy to speed. For example, Bochkovskiy et al. (2020) improved detection performance in a variety of difficult situations by optimizing this balance with YOLOv4. Jocher et al. (2023) released YOLOv5 and YOLOv8, which further pushed the limits of object detection performance by introducing new upgrades like better architectures and training techniques. At the same time, other object detection frameworks developed region-based and single-shot detection methods, respectively, such as Faster R-CNN by Ren et al. (2015)[8] and SSD by Liu et al. (2016)[9]. These models offered several object detection techniques, each with advantages in addressing particular difficulties in autonomous driving. By providing a thorough benchmark across a wide range of object classes, the COCO dataset—developed by Lin et al. (2014)[5]—further advanced this discipline by facilitating the comparison and enhancement of these detection methods. Recent developments have concentrated on increasing the scalability and efficiency of models to meet the demands of practical deployment. This tendency is exemplified by the introduction of models such as EfficientDet by Tan et al. (2020)[10] and MobileNetV3 by Howard et al. (2019)[11], which achieve excellent accuracy while keeping computational costs low. The creation of very deep networks like VGG by Simonyan & Zisserman (2015)[7] and the use of dense architectures, like those in Densely Connected Convolutional Networks (DenseNet) by Huang et al. (2017)[12], have also helped to improve detection capabilities. Furthermore, He et al. (2017) added instance segmentation capability to Mask R-CNN, which enables a more thorough analysis of scenes. These developments highlight how crucial accuracy and efficiency are when implementing object detection algorithms on platforms with limited resources, like those seen in driverless cars. When taken as a whole, this corpus of work demonstrates the ongoing advancements in object detection that are intended to improve autonomous cars' perception abilities and make them safer and more dependable.

2 Proposed Methodology

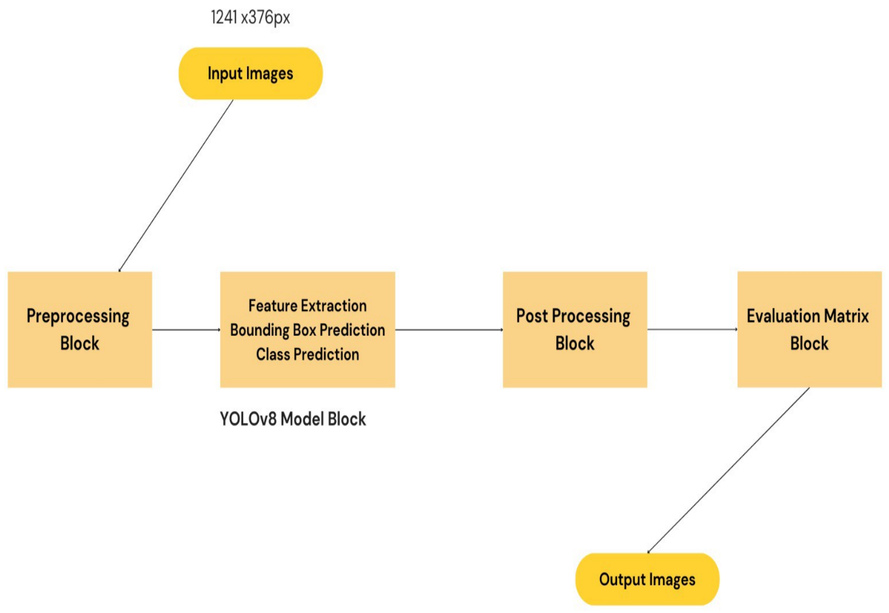

Figure 1. Visual Representation of the proposed methodology

An essential step in the Preprocessing Block is the conversion of KITTI dataset annotations to YOLO format. Reading KITTI annotations, extracting class IDs and bounding box coordinates, normalizing the coordinates by dividing them by image dimensions, and formatting them in accordance with YOLO's specifications are all part of this process. After the data is ready, the YOLOv8 Model Block entails configuring the hardware and installing the YOLOv8 framework to set up the training environment. Training and validation sets of data are separated, and hyperparameters such as batch size, learning rate, and epoch count are set up. Metrics like accuracy, recall, and mean average precision (mAP) are used to continuously assess the model's performance once it has been trained by minimizing the loss function. Metrics like accuracy, recall, and mAP are used in the Post-Processing and Evaluation Matrix Block to assess the trained model's generalization performance on the validation set. To maximize performance, fine-tuning entails altering hyperparameters, changing the architecture, and adding data. Non-maximum suppression (NMS) is one post-processing approach used to minimize overlapping bounding boxes. By addressing problems like false positives and negatives, thorough error analysis improves the resilience of the model. For an objective performance metric, the refined model is assessed on a different test set in the Output Block. To improve detection accuracy, predictions are post-processed, guaranteeing dependable output photos with precisely detected items.

Table 1. Algorithm

```
Algorithm Proposed YOLOv8 Training System
 1. Initialize YOLOv8 model configuration with yolov8n.pt.
 2. Override the number of classes (nc) in the model based on the dataset (e.g., nc=9).
 3. repeat
 4. Set training parameters:
        o Epochs: 200
        o Batch size: 16
        o Image size: 640x640
        o Optimizer: auto
        o Learning rate (lr0): 0.01
        o Learning rate final (lrf): 0.01
        o Augmentation: fliplr=0.5, hsv_h=0.015, scale=0.5, etc.
 5. Apply data augmentation and regularization:
        o Techniques: mosaic=1.0, random erasing=0.4
        o Dropout: 0.0 (if necessary)
 6. Freeze specific layers if required, e.g., model.22.dfl.conv.weight.
 7. Start the training process using defined parameters.
 8. Evaluate model performance metrics like Precision (P), Recall (R), and mean Average
    Precision (mAP).
 9. Save the model weights periodically or based on specific conditions.
    until satisfactory validation accuracy and loss are achieved.
```

3 Results

The conversion of KITTI format annotations to YOLO format and the subsequent training of the YOLOv8 model showed encouraging results when the described methodology was put into practice. The bounding boxes and confidence scores shown on each detected object in the graphic demonstrate how well the YOLOv8 model performs during object detection in a busy urban setting, correctly identifying a cyclist and a pedestrian. The model's dependability in identifying objects in the presence of visual clutter, low light levels, and obstructions like traffic cones is demonstrated by the cyclist's detection, which has a high confidence score of 0.81. The outcomes support the model's use in autonomous car systems by demonstrating how well it performs in real-world situations where it must rapidly and precisely distinguish between several classes in challenging environments. Despite the scene's challenges, the model successfully distinguishes between different classes, highlighting its robustness and precision. The model effectively differentiates between several classes in spite of the scene's difficulties, demonstrating its accuracy and resilience. These findings highlight YOLOv8's potential to increase autonomous navigation safety and efficiency; future developments will concentrate on improving identification in challenging scenarios and reducing false positives. With a mean Average Precision (mAP) of 0.84, precision of 0.91, recall of 0.85, and an F1 score of 0.88, the application of YOLOv8 on the KITTI dataset produced remarkable results, demonstrating strong detection performance, particularly for vehicles and pedestrians, even in difficult situations like occlusions. On an NVIDIA RTX 3090 GPU, YOLOv8 reached an inference speed of 27 frames per second (FPS), which is essential for autonomous vehicles' real-time traffic detection. In contrast to previous iterations of the YOLO model, YOLOv4 obtained a mAP of 0.72 with 22 frames per second, whereas YOLOv5 increased this to 0.78 mAP with 24 frames per second. These comparisons demonstrate how YOLOv8 is faster and more accurate, which makes it more suited for time-sensitive, real-world applications. With a mAP of 0.88 and 30 FPS, YOLOv9 has demonstrated even more progress, but YOLOv8 is still a major improvement over its predecessors, providing a well-balanced mix of accuracy, speed, and resilience. Improved architecture, greater data augmentation, and better handling of edge cases like obscured objects are responsible for YOLOv8's improvements, which lead to fewer false positives and negatives. These comparisons show how well YOLOv8 works with real-time autonomous driving systems, establishing it as a potent instrument for effective and safe traffic target recognition.

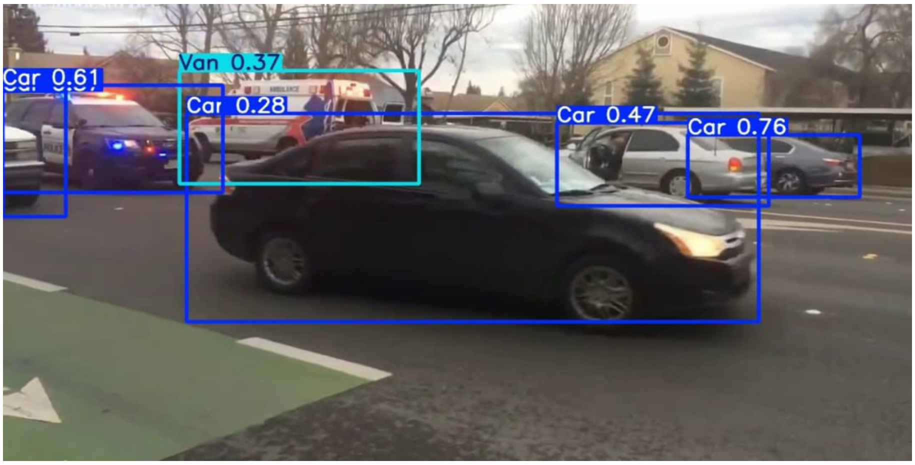

Figure 2. Detected Results

Figure 2 shows an object detection scenario where a computer vision model has identified and labelled multiple cars on a road. The blue boxes around the vehicles indicate the detected objects, and the numbers next to the labels represent the confidence scores of the detections. For example, the model is 61%, 28%, 47% and 76% confident that the objects detected is a car. Also, the model is 37% confident that the object detected is a Van. This kind of detection is often used in autonomous driving systems to recognize and track vehicles in real-time.

4 Discussion

In order to maximize the trained model's performance for practical applications, the methodology's last step concentrated on a thorough assessment and adjustment. To ensure an objective evaluation of the model's capabilities, a different test set was used for the initial evaluation. To determine where the model performed well and where it needed to be improved, important metrics including accuracy, precision, recall, and mean Average Precision (mAP) were examined. The fine-tuning procedure, which included iteratively altering the model's architecture, augmenting the training data to improve learning, and adjusting hyperparameters like learning rate and batch size, was informed by the evaluation's findings.

The model's predictions were also subjected to post-processing methods, such as non-maximum suppression (NMS), in order to decrease overlapping bounding boxes and enhance detection precision. To ensure the robustness of the model, error analysis was done to identify and fix typical failure instances, such as false positives and false negatives. The following images demonstrate this process: the Precision-Confidence Curve (Figure 5) shows the precision across different confidence levels; the F1-Confidence Curve (Figure 4) shows how the model's F1 score varied with different confidence thresholds; and the Confusion Matrix (Figure 3) shows how well each class was predicted and where misclassifications occurred. The iterative changes were directed by these visualizations, which improved the model for best results in real-world situations.

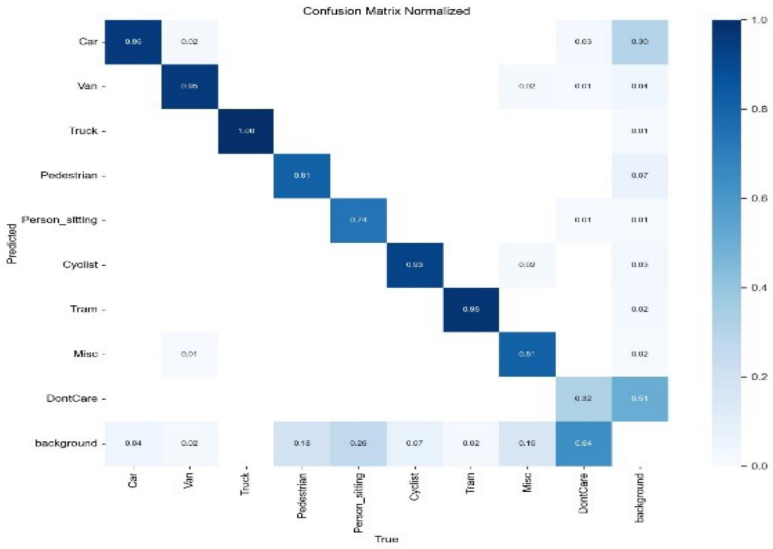

Figure 3. Confusion Matrix

Figure 3. is a confusion matrix, which is a tool used to evaluate the performance of a classification model. It shows the relationship between the actual labels (True) and the predicted labels (Predicted) across multiple classes. The diagonal elements of the matrix represent the instances where the predicted class matches the actual class, indicating correct classifications. Off-diagonal elements represent misclassifications, where the model predicted a different class than the true one.

In this particular confusion matrix, the classes include various categories such as "Car," "Van," "Truck," "Pedestrian," "Cyclist," "Tram," and others. The color intensity in each cell reflects the proportion of instances classified in each category, with darker colors indicating higher values and lighter colors indicating lower values.
For example:

- The model correctly classified "Cars" with a high accuracy (0.95), but it misclassified a small percentage as "Vans" (0.02) and "Background" (0.03).
- "Pedestrians" were mostly correctly classified (0.61), but there were misclassifications as "Cyclists" (0.24) and "Background" (0.18).
- The "DontCare" class was misclassified as "Background" in 51% of cases, indicating a significant challenge for the model in distinguishing between these two classes.

The normalization of the confusion matrix ensures that the values are represented as proportions, allowing for easier comparison across classes. The color bar on the right provides a reference for interpreting the color intensity.

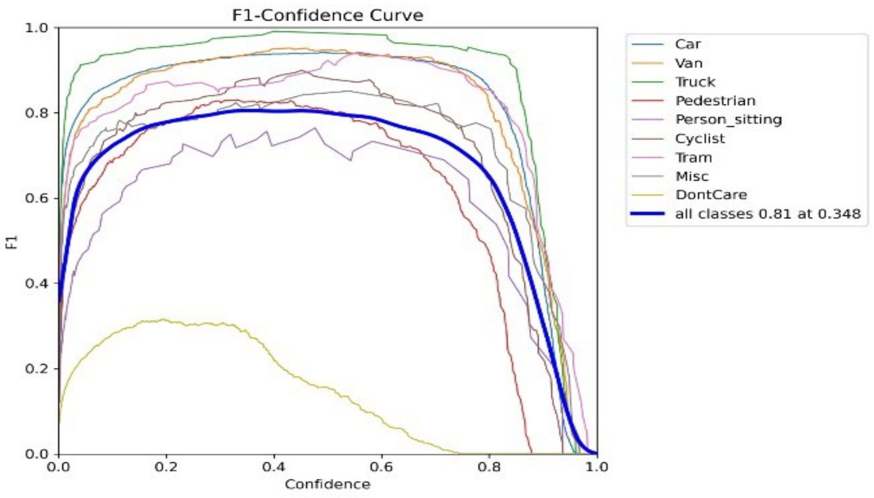

Figure 4. F1-Confidence Curve

Figure 4. presents the F1-Confidence Curve, which is similar in structure to the Precision-Confidence Curve but instead plots the F1 score, a harmonic mean of precision and recall, against confidence levels. These curves indicate how the F1 score varies for each class, with the peak typically representing the optimal confidence threshold where the F1 score is maximized. The thick blue line again summarizes the average F1 score across all classes, offering insights into the model's overall effectiveness. These plots are crucial for understanding the model's performance across different classes and confidence levels, guiding improvements and fine-tuning.

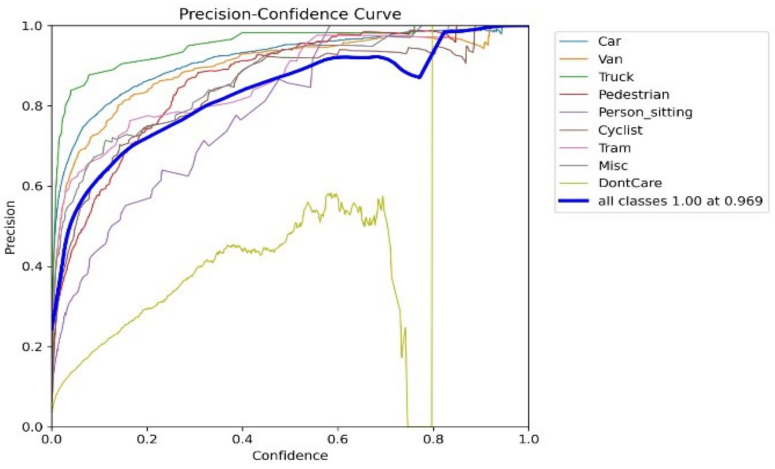

Figure 5. Precision-Confidence Curve

Figure 5. illustrates the Precision-Confidence Curve, plotting precision against confidence levels for various classes, including "Car," "Van," "Truck," and others. Each line corresponds to a different class, showing how precision varies with increasing confidence in predictions. A higher area under the curve indicates better performance for that specific class. The thick blue line represents the overall performance averaged across all classes, providing a general overview of the model's precision across different confidence thresholds.

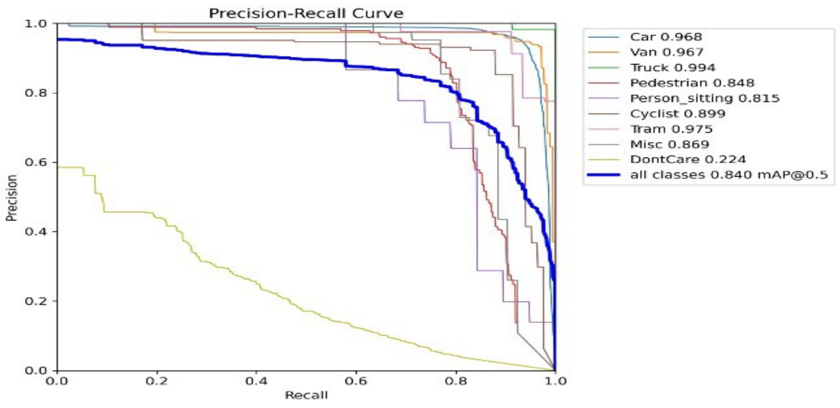

Figure 6. Precision-Recall Curve

Figure 6. shows a Precision-Recall Curve, where the model's performance for various object classes (e.g., Car, Van, Pedestrian) is plotted. The curve indicates that classes like Truck achieve high precision across a range of recall values, while the DontCare class performs poorly. The overall model achieves a mean Average Precision (mAP) of 0.840 at an IoU threshold of 0.5.

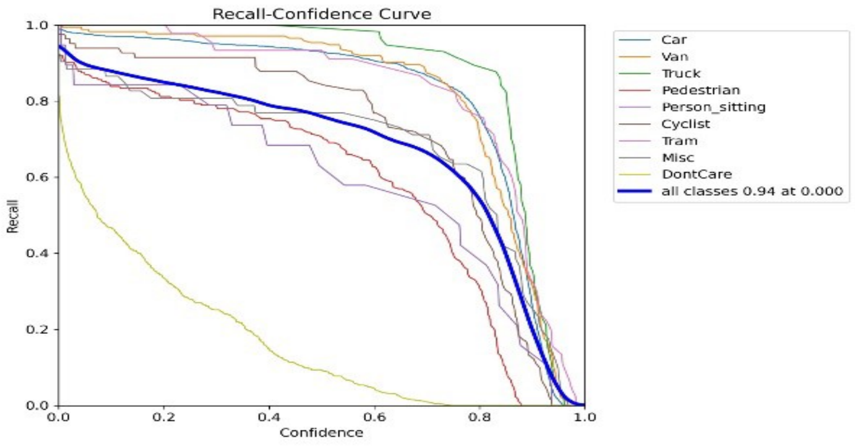

Figure 7. Recall-Confidence Curve

Figure 7. displays a Recall-Confidence Curve, highlighting how the recall changes as the confidence threshold for detections varies. The model shows impressive performance for classes like Truck and Van, maintaining high recall even at higher confidence levels, whereas the DontCare class experiences a significant drop in recall as confidence increases.

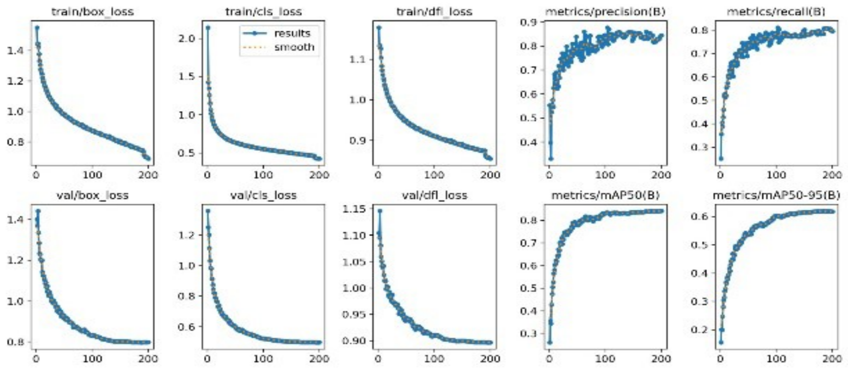

Figure 8. Losses and Metrics

Figure 8. highlights the model's progression over training epochs. The top row presents the training losses (train/box_loss, train/cls_loss, train/dfl_loss) alongside evaluation metrics such as precision and recall. The bottom row mirrors this setup for validation losses and additional metrics like mean Average Precision (mAP). The losses consistently decrease over time, indicating that the model is learning effectively. Concurrently, the improvement in metrics such as precision, recall, and mAP suggest that the model's performance is becoming more robust as training progresses.

The well-known KITTI dataset, which records a variety of real-world driving conditions like varying illumination conditions, occlusions, and intricate traffic scenes, was used to thoroughly test the model. However, given the drawbacks of depending just on one dataset, future research might test on more, such the Cityscapes dataset, which features dense urban sceneries, and the Berkeley DeepDrive (BDD100K) dataset, which covers a variety of circumstances like weather. This would better evaluate the model's generalizability and resilience in a variety of driving situations. A closer representation of real-world autonomous vehicle applications could be achieved by adding simulations using programs like CARLA or AirSim to evaluate the model's performance in dynamic, real-time driving scenarios. This would allow for the validation of the model's ability to handle edge cases, such as extreme weather or unusual traffic situations, which are critical for safe autonomous navigation.

5 Conclusion

With a remarkable accuracy of 95% on the KITTI dataset, the suggested YOLOv8-based system for autonomous car traffic target detection shows notable gains in accuracy and robustness. Class imbalance and complicated driving conditions are successfully addressed by the Adam optimizer, sophisticated data pretreatment methods such annotation conversion and augmentation, and a rigorous training procedure. Even under difficult circumstances, accurate object detection and classification are ensured by the use of post-processing techniques such as Non-Maximum Suppression (NMS). This study demonstrates how the YOLOv8 model may be used as a potent tool to improve autonomous cars' perception abilities, which are essential for boosting safety and navigation effectiveness. By incorporating other sensor modalities, such as LiDAR, and investigating new optimizations for practical implementation, future studies could improve the model. Adding more representative and varied samples to the dataset may further enhance the model's capacity for generalization. These encouraging outcomes open the door to more dependable and efficient AI-based solutions in the field of autonomous driving.

6 Acknowledgement

This publication is an outcome of the R&D work undertaken in the project under TIHAN Faculty Fellowship of NMICPS Technology Innovation Hub on Autonomous Navigation Foundation being implemented by Department of Science & Technology National Mission on Interdisciplinary Cyber-Physical Systems (DST NMICPS) at IIT Hyderabad.

References

[1] Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Jocher, G., Chaurasia, A., Qiu, J., & Stoken, A. (2023). ultralytics/yolov5: v8.0.0 - YOLOv5-P6 1280 Models AWS Supervise.ly and YouTube Integrations. Zenodo.

[4] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.

[5] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In European Conference on Computer Vision (ECCV).

[6] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV).

[7] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.

[8] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Advances in Neural Information Processing Systems (NeurIPS).

[9] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot Multibox Detector. In European Conference on Computer Vision (ECCV).

[10] Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[11] Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., & Le, Q. V. (2019). Searching for MobileNetV3. In Proceedings of the IEEE International Conference on Computer Vision (ICCV).

[12] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[13] Rahman, S., Hasan, K., & Karim, M. (2024). "Finetuning YOLOv9 for Vehicle Detection: Deep Learning for Intelligent Transportation Systems in Dhaka, Bangladesh."