

Application of Computer Vision for Detecting Electric Scooters in Vehicle Blind Spots

Milan Stojanović

Abstract – Due to the increasing use of electric scooters as a means of transportation in cities, ensuring the safety of their riders is very important. Limited visibility and the presence of blind spots in large vehicles, especially trucks and buses, and the difficulty of spotting electric scooters puts riders in dangerous situations. In this paper, we will demonstrate a computer vision approach for detecting electric scooters and present an evaluation of its performance. The idea is to create a system which can be implemented in large vehicles using dedicated cameras and software to warn drivers when an electric scooter is next to or behind the vehicle. The use of computer vision for this purpose is more effective than the use of simple sensors because this method accurately detects if there is an electric scooter, and not some other obstacle.

Keywords – Computer vision, Electric Scooters, Blind Spots, Traffic Safety.

I. INTRODUCTION

As traffic congestion in cities increases due to the growing number of cars, electric scooters are becoming more popular as a mode of transport, because they allow easier movement through urban areas and people no more have problems with parking. The increase in the number of electric scooters on city streets also introduces new challenges for riders safety, because by law these scooters must travel on the roadway in parts of the city where there are no bicycle lanes or paths. Since these scooters are small in size, they are often hard for drivers to notice, especially for drivers of large vehicles such as trucks, delivery vans, or buses, particularly when the scooter moves into the blind spot of these vehicles. Such situations significantly increase the risk of accidents, so it is necessary to find a solution to prevent them [1].

Modern vehicles today are equipped with electronic systems and sensors for various purposes. Sensors used to detect objects around the vehicle are crucial. However, they are often limited, as they can detect objects but cannot identify what object that is. Of course, any contact between a vehicle and a physical object is undesirable, but it becomes particularly dangerous if the object is, for example, a moving electric scooter. This exposes a key limitation of conventional sensor systems, which cannot adequately inform the driver about what is present in the blind spot of their vehicle.

Milan Stojanović, Miljana Milić, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: milancestojanovic@elfak.rs, miljana.milic@elfak.ni.ac.rs

Computer vision offers a solution to this problem. Using cameras and advanced detection algorithms, these systems can reliably identify obstacles. For example, pedestrian detection in front of moving vehicles is becoming more and more common in modern “smart” cars to enhance safety.

In this paper, we will show how electric scooters can be detected using computer vision, with a special focus on the potential future application of systems for detecting electric scooters in vehicle blind spots.

Chapter II will provide an overview of the Computer Vision, with a particular focus on the YOLO framework for object detection. Chapter III will describe the creation of the dataset, Chapter IV will present training of the model, while Chapter V will cover model evaluation and testing.

II. COMPUTER VISION TASKS AND YOLO FRAMEWORK

Computer vision is a field of artificial intelligence that allows computers and machines to analyze and interpret visual information from images and video [2]. It uses machine learning to extract information from images and videos, enabling computers to understand them in a way similar to humans. As camera technology improves, capturing high resolution images, computer vision has become a very useful tool, with applications in autonomous vehicles and smart car systems, video surveillance, medicine, and many other areas.

Computer vision algorithms can be trained for various tasks, such as:

- classification,
- object detection,
- segmentation,
- object tracking,
- face recognition,
- pose estimation.

Image classification is used to determine what is present in an image without providing information about the position of a specific object (e.g., it classifies the entire image as “dog” if a dog appears in it). Object detection locates objects in an image using bounding boxes, showing both what and where the objects are (e.g., detecting pedestrians in a security camera frame; analyzing each frame independently). The task of segmentation allows precise separation of objects at the pixel level (dividing an image into regions belonging to different classes or

objects). In semantic segmentation, each pixel is assigned a class, but individual objects of the same class are not separated (e.g., all pixels labeled as “road” are green, all pixels labeled as “vehicle” are red, without separating individual cars). In instance segmentation, each pixel is assigned to a specific object (e.g., each car is marked with a different color). Object tracking follows the same object across multiple video frames by assigning it a unique ID. This allows the object’s movement and changes in position to be monitored over time. (e.g., tracking a car’s trajectory in highway footage). This task is based on object detection but focuses on analyzing object motion throughout a video sequence. Pose estimation is used to analyze the position and orientation of the human body in a video.

In this work, the object detection task is employed to identify and locate electric scooters in images and video frames, providing both the class of the detected object and its position within the scene [3].

YOLO (You Only Look Once) is an object detection method that combines object classification and localization into a single step [4]. Unlike traditional approaches that analyze different regions of an image separately by first proposing potential objects and then classifying them, YOLO processes the entire image at once and predicts all objects simultaneously. It provides information about which objects are present in the image as well as their positions using bounding boxes around each detected object. In the background, YOLO is based on supervised deep learning and convolutional neural networks (CNN) [5], which learn visual patterns in the image directly from annotated data and enable efficient, real-time object detection.

The main advantages of YOLO include its high speed, which allows it to operate in real time and makes it suitable for electronic systems in vehicles. In addition, YOLO is efficient, as it can detect multiple objects simultaneously, including small and fast moving objects such as electric scooters. Another important advantage is its scalability, since multiple model versions (n, s, m, l, x) are available, offering different compromises between speed and accuracy.

YOLO is particularly suitable for use in vehicles because it enables accurate detection and tracking of moving objects in complex urban environments. For practical use of YOLO models in actual systems, the Ultralytics framework is commonly used, as it simplifies model training, automates image and video processing, bounding box prediction, and result filtering [6]. Using this framework, YOLO models can be efficiently implemented and adapted to specific object detection tasks.

III. CREATING THE DATASET

This chapter describes the preparation of dataset and the training of a YOLO model for detecting electric scooters. The dataset consists of images captured from different angles, with different lighting conditions, allowing the

model to learn to detect objects in real situations.. Each image is annotated with the objects classes and precise locations using bounding boxes, providing supervised learning for object detection and localization. During training, the model adjusts its weights to maximize accuracy in identifying and locating objects in both images and video.

Images in the dataset were annotated using Label Studio, a software tool designed for various computer vision tasks. In this case, objects were annotated using bounding boxes, which is essential for detecting electric scooters. The dataset contains images taken from different angles and with different quality to ensure the model can detect scooters in real-world conditions. Each image was carefully reviewed, and electric scooters were manually labeled to provide precise object locations and classes. This process guarantees accurate and consistent annotations, which are critical for effective YOLO training and reliable detection in practical scenarios. Figure 1 shows the Label Studio screen with loaded images from the dataset.

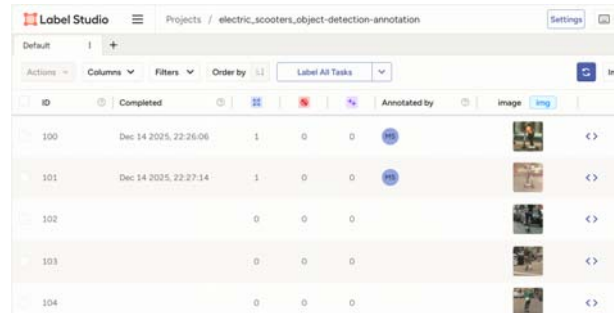


Fig. 1. Label Studio screen with loaded images.

Figure 2 shows example of annotating electric scooter image. It was necessary to create a label electric_scooters that refers to objects that are labeled using bounding boxes.



Fig. 2. Example of annotating the electric scooter in image.

After successfully annotating the electric scooters in all images, Label Studio allows exporting the image dataset along with the labels in a format suitable for use with YOLO. For each image in the dataset, a corresponding .txt file was generated containing information about the object in YOLO format, as shown in Figure 3. The file contains a single line representing one object, which includes the object class ID (in this case, 0 = electric_scooter) along with four numbers defining the bounding box: the x-coordinate of the object center (x_center), the y-coordinate of the center (y_center), the width (width), and the height (height). All values are normalized between 0 and 1 relative to the image dimensions. This format allows the model to learn the precise location and size of objects in the image.

```
0 0.5375 0.4140625 0.32734375 0.828125
```

Fig. 3. Example of a YOLO annotation file for an image in the dataset.

The acquired data, which contains 300 images, must be split into training, validation, and test sets in the proportions of 70%, 15%, and 15%, respectively. The training set is used to train the model and adjusting the weights of the YOLO neural network based on the images and annotations. The validation set allows monitoring the model's performance during training and adjusting hyperparameters, which helps reduce the risk of overfitting. The test set contains images the model has not seen before and is used for the final evaluation of its accuracy, precision, and ability to generalize to real-world scenarios.

III. MODEL TRAINING

The training process for an object detection model typically involves three main stages: training, validation, and prediction. During the training stage, the model learns to recognize patterns in the images from the training dataset, updating its parameters to improve detection accuracy. The validation stage evaluates the model on a separate set of images reserved for validation, providing feedback on how well the model generalizes to new data and helping to prevent overfitting. The predict stage, also called inference, demonstrates how the trained model can be applied to new images or videos to detect objects in real time.

To train any object detection model using the Ultralytics package, it is necessary to create a .yaml file. This file serves to describe the dataset used during model training. It does not contain the images or annotations themselves, but rather information about where they are located and which object classes the model should recognize. Typically, a .yaml file includes paths to the dataset subsets (train, validation and test), number of classes and classes names. The number of classes represents the total number of different object types that the model

should detect in the dataset, while the class names correspond to the labels used during annotation of objects in the images. In this example, since the model detects only electric scooters, the number of classes is 1, and the class name is electric_scooter. The example of a .yaml file is shown in Figure 4.

```
train: ../dataset/train/images
val: ../dataset/valid/images
test: ../dataset/test/images

nc: 1
names: ['electric_scooter']
```

Fig. 4. Example of a .yaml file.

In Ultralytics, the mode specifies the operation the model should perform. For instance, train mode starts the training process, predict mode runs inference on new images or videos, and other modes allow exporting or evaluating the model. This flexible approach makes it easy to control exactly what the model does at each stage.

The training script for the object detection model should be written in a .py file within the same project. The script imports YOLO from the Ultralytics package, which is used to load the model. In this case, the YOLO 11 Nano model is chosen, a lightweight object detection model suitable for fast training and inference. Ultralytics supports a range of YOLO versions, from 7 up to 12, with each successive version offering improvements in speed, accuracy, and overall performance.

YOLO 11 is the best choice for this application because it is considered one of the best models for object detection, providing an excellent balance between high detection accuracy and computational efficiency, which is important when working with limited resources or deploying the model on embedded devices.

The train mode in the YOLO environment supports various arguments that allow precise control over the model training process. Some of the most important arguments include the data argument defines the path to a previously created .yaml file describing the dataset, epochs specifying the number of training epochs, batch represents the number of images processed simultaneously in one training step, directly affecting training stability, speed, and GPU memory usage; imgsz determines the input image size, which is especially important when deploying the model on embedded devices, device specifies whether training is performed on CPU or GPU, with GPUs being significantly faster and more efficient for parallel processing of large amounts of data compared to CPUs, workers sets the number of CPU threads for data preparation, ensuring a continuous flow of batches to the GPU during training. In this example, the model is trained using the GPU (device 0) with a batch size of 16, a single worker thread, 120 epochs, and an input image size of 640.

Executing the training script loads the YOLO model and begins iterating through the dataset according to the specified arguments. During training, a log output is displayed in the console for each epoch, providing detailed metrics that allow monitoring of the model's learning progress. These metrics, which include losses for bounding box regression, classification, and other components, as well as performance indicators such as precision, recall, and mean average precision (mAP), change over the course of epochs: typically, loss values decrease as the model learns, while precision, recall, and mAP gradually increase, reflecting improved detection performance. Examining this output makes it possible to track model improvement over time, identify potential issues like underfitting or overfitting, and adjust hyperparameters if necessary. The values for some characteristic training epochs are shown in Figure 5.

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
1/120	4.96G	1.278	1.94	1.524	35	640	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95)
all		240	256	0.555	0.609	0.607	0.35
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
40/120	5.16G	0.8176	0.6614	1.189	38	640	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95)
all		240	256	0.889	0.936	0.946	0.767
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
80/120	5.16G	0.6934	0.5319	1.102	45	640	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95)
all		240	256	0.895	0.962	0.95	0.823
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
120/120	5.16G	0.3973	0.3089	0.9716	18	640	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95)
all		240	256	0.898	0.965	0.954	0.827

Fig. 5. Training values for some characteristic training epochs.

During training, a trend of improving performance can be observed over the epochs. Loss values (including box loss, classification loss, and DFL loss) gradually decrease, indicating that the model is better adjusting its weights and reducing errors in object localization and classification. At the same time, performance metrics such as precision (P), recall (R), and mean average precision (mAP50 and mAP50-95) increase, showing that the model becomes more accurate and reliable in detecting and classifying objects. GPU memory usage remains stable, suggesting that parameters like batch size and number of workers are well optimized. These trends collectively demonstrate that the model is progressively learning patterns from the training dataset and that the training process is successful.

When training completes, YOLO generates two main model weight files: best.pt and last.pt. The best.pt file contains the weights from the epoch where the model achieved the highest performance according to the chosen evaluation metric (usually mAP), representing the most accurate version of the model. The last.pt file contains the weights from the final epoch of training.

Figure 6 shows the loss charts, tracking Box, CLS, and DFL losses during training and validation. Box loss measures errors in predicted bounding boxes, CLS loss shows errors in class predictions, and DFL loss improves bounding box accuracy. Figure 7 shows performance

metrics including precision, recall, and mAP, which reflect detection quality. Precision is the proportion of correct detections, recall is the proportion of true objects detected, mAP50 shows accuracy at 50 percent overlap, and mAP50 95 averages accuracy across overlap thresholds from 50 to 95 percent.

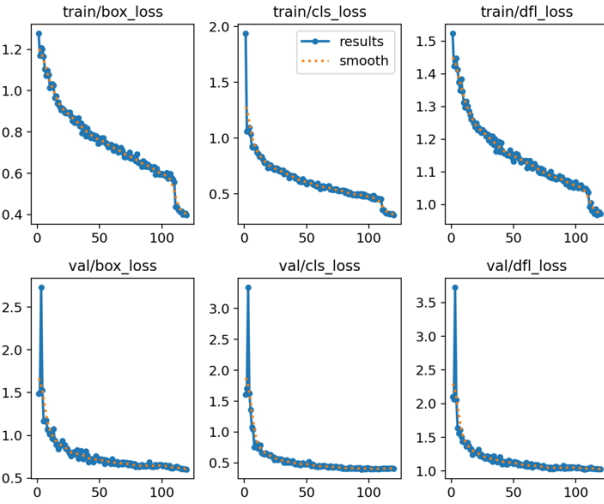


Fig. 6. Training and validation loss graphs.

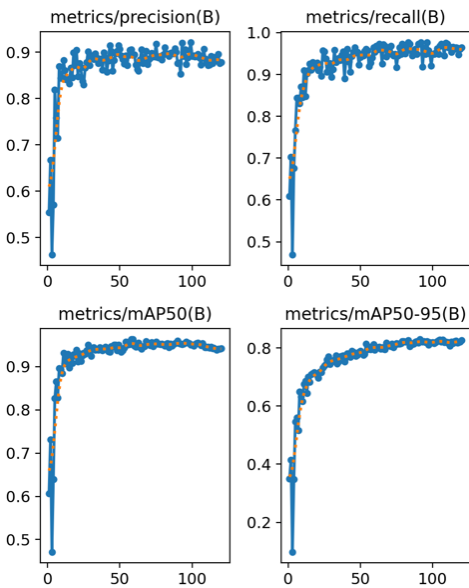


Fig. 7. Model performance metric graphs.

The precision-recall curve in Figure 8 illustrates the trade-off between precision and recall at different prediction confidence levels, providing insight into the model's overall detection performance. Points higher on the curve indicate that the model is able to detect a larger proportion of true objects while keeping false positives low. The curve also highlights the confidence thresholds at

which the model achieves its best balance between detecting objects accurately and minimizing incorrect detections.

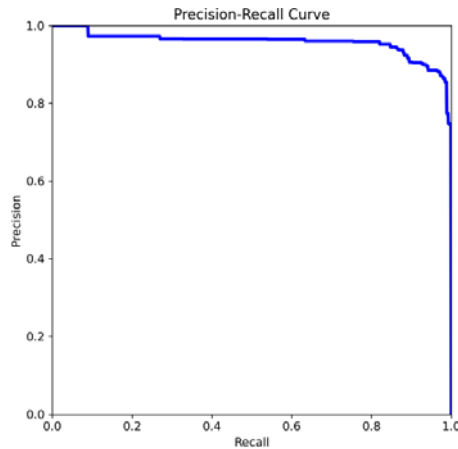


Fig. 8. Precision-Recall curve.

V. MODEL EVALUATION AND TESTING

After training, the model's performance is evaluated using videos from real-world test scenarios. During the evaluation and testing phase, the trained YOLO model is applied to video to evaluate how well it detects objects in motion. Each frame of the video is processed by the model, providing bounding boxes, class labels, and prediction confidence values for detected objects. Prediction confidence represents the model's certainty for each detection and is displayed next to the bounding box on the video frame, allowing quick visual assessment of detection reliability. In this study, confidence values between 0.6 and 0.7 are considered acceptable, 0.7 to 0.85 are good, and values above 0.9 are very good, though rarely achieved without a large dataset. Figures 9 and 10 show examples of detection on video frames, captured at the moment when objects were detected, illustrating both the bounding boxes and the corresponding prediction confidence.



Fig. 9. Example of electric scooter detection on a video frame.

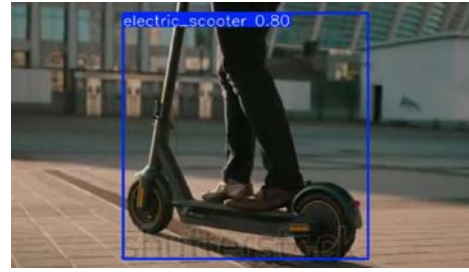


Fig. 10. Example of electric scooter detection on a video frame.

These examples illustrate that the model can maintain consistent detection performance across frames with varying prediction confidence. While higher confidence values indicate more certain detections, slightly lower values are still within acceptable limits and reflect realistic variations in object appearance, lighting, and movement.

VI. CONCLUSION

This paper demonstrates an efficient approach to detecting electric scooters using computer vision. Based on 240 processed train images from the dataset, it was determined that the model can be trained to effectively detect electric scooters, which is particularly important for identifying these vehicles in the blind spots of large vehicles, trucks, and buses. The model's performance could be further improved if a larger dataset were used for training. Implementing this model on a hardware platform installed in large vehicles could significantly reduce traffic accidents involving electric scooter riders.

REFERENCES

- [1] C. Cloud, S. Heß, and J. Kasinger, "Do shared e-scooter services cause traffic accidents? Evidence from six European countries," *arXiv preprint arXiv:2209.06870*, Sep. 2022.
- [2] A. R. Pathak, C. K. Patel, V. R. Patel, S. Jambhorkar, S. S. Lahane, and A. A. Ghatol, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103812, Jan. 2023.
- [3] J. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep Learning for Generic Object Detection: A Survey," *International Journal of Computer Vision*, vol. 128, pp. 261–318, Feb. 2020.
- [4] R. Kaur, S. Singh, "A Review of YOLO Object Detection Based on Deep Learning," *Frontiers in Computing and Intelligent Systems*, Jan. 2023.
- [5] M. Wang and W. Leelapatra, "A Review of Object Detection Based on Convolutional Neural Networks and Deep Learning," *International Scientific Journal of Engineering and Technology (ISJET)*, vol. 6, no. 1, pp. 1–7, 2022.
- [6] Ultralytics, "Ultralytics Official Website," 2025. [Online]. Available: <https://www.ultralytics.com/>