Vehicle Detection and Classification via Yolov8: A Comprehensive Survey

Asadollah Shahbahrami

E-mail:shahbahrami1@yahoo.com

Bahar Khouban, Fereshteh Namazi

E-mail:baharkhuban@gmail.com, fereshte.namazi.edu@gmail.com

Bahareh Hosseini, Zahra Keyhan Pour

E-mail:Bahare.hsn8@gmail.com, keyhanzara@gmail.com

Abtin Hossein Nezhad

Abtin.hosseinnezhad121@gmail.com

Abstract: The ever-increasing number of vehicles on the road necessitates efficient traffic monitoring systems for urban planning and congestion management. This report discusses an approach for vehicle detection and classification using the YOLOv8 deep learning algorithm, focusing on classifying vehicles into relevant categories for traffic analysis, including heavy vehicles (buses and trucks) and light vehicles (cars and motorcycles). The program localizes and classifies vehicles in traffic scenes, making it suitable for practical traffic monitoring applications. The report explores the training process of the model on a custom dataset specifically designed for traffic analysis and evaluates the program's performance in terms of detection and classification. The findings demonstrate the effectiveness of this approach in vehicle detection and classification for traffic analysis, paving the way for its integration into intelligent traffic monitoring systems and ultimately contributing to improved traffic management and urban planning.

Keywords: Vehicle detection, Vehicle classification, YOLOv8, labelling.

Reference to this paper should be made as follows: Shahbahrami, A., Khouban, B., Namazi, F., Hosseini, B., Keyhan Pour, Z., & Hossein Nezhad, A. (2024). Vehicle Detection and Classification via Yolov8: A Comprehensive Survey.

Biographical notes: Asadollah Shahbahrami received his BSc and MSc in Computer Engineering from Iran University of Science and Technology and Shiraz University in 1993 and 1996, respectively. He obtained his PhD degree from the Delft University of Technology, The Netherlands in 2008. He has been working at the University of Guilan since August 1996. He is a Professor in the Department of Computer Engineering at the University of Guilan. His research interests include advanced computer architecture, image and

video processing, multimedia instructions set design, reconfigurable computing, parallel processing, and SIMD programming.
Bahar Khouban, Fereshteh Namazi, Bahareh Hosseini, Zahra Keyhan Pour, and Abtin Hossein Nezhad are all Computer Engineering bachelor students.

1 Introduction

The ever-increasing volume of vehicles on urban roads necessitates the development of more efficient traffic monitoring systems. These systems play a crucial role in informing urban planning strategies and facilitating traffic congestion management. Fortunately, advancements in machine learning and computer vision have enabled the creation of sophisticated solutions for vehicle detection and classification. This report explores the application of the YOLOv8 deep learning algorithm to address this growing challenge, while acknowledging the inherent challenges associated with real-world traffic monitoring.

YOLOv8, recognized for its robust object detection capabilities, offers an effective solution for real-time traffic monitoring. The discussed method leverages YOLOv8's ability to localize and classify vehicles within diverse traffic scenarios. This facilitates the categorization of vehicles into predefined classes, such as heavy (buses and trucks) and light (cars and motorcycles). This classification holds significant value for various practical applications, including autonomous vehicles, traffic control systems, restricted area enforcement (where heavy vehicles are prohibited), and dynamic traffic light control based on real-time traffic conditions. Furthermore, integrating this system into existing traffic management infrastructure can enhance the effectiveness of speed control cameras, traffic law enforcement, accident monitoring, and driving penalty management.

Our approach prioritizes the system's real-time processing capabilities. To ensure the model's applicability and effectiveness in real-world scenarios, we trained the YOLOv8 model on a custom dataset meticulously designed for traffic analysis.

Real-world traffic monitoring systems face inherent challenges due to factors like variations in lighting, object appearance, occlusion, and background clutter (2) . Variations in lighting, particularly low-light conditions, can significantly impact detection. Vehicles come in a wide variety of shapes and sizes, and their pose can affect how the model perceives them. Vehicles can be occluded by other objects, and complex backgrounds can introduce noise that can confuse the model.

Research Questions:

This research investigates the following questions to enhance vehicle detection and classification for real-world traffic monitoring applications:

RQ1: How does YOLOv8 enhance the efficiency and precision of object detection compared to its predecessors?

RQ2: What are the advantages of Roboflow over Label Studio for computer vision annotators?

RQ3: How does the diversity and quality of collected data influence the performance and robustness of a vehicle detection model in real-world applications?

By addressing these research questions, this study aims to contribute to the advancement of vehicle detection and classification for improved traffic monitoring systems. The report will delve into the details of the YOLOv8 training process, present the methodology for dataset preparation that specifically addresses these challenges, and discuss the system's

performance under various conditions. The anticipated outcome is to demonstrate the practicality and efficiency of YOLOv8, paving the way for its widespread adoption in various traffic monitoring and control systems.

2 Previous Studies

After several years, a Convolutional Neural Network (ConvNet/CNN) (27) was introduced. A ConvNet is a deep learning algorithm that can receive an input picture, allocate significance parameters and biases to numerous aspects or objects in the picture, and be capable of distinguishing one from the other. It involves less pre-processing than other classification algorithms and, with adequate training, can learn filters and characteristics over time, unlike traditional methods. This approach, inspired by the linked form of neurons in the human brain, decreases the computational power necessary to process data, making it faster than traditional methods (26).

The importance of vehicle detection and classification in traffic management is well-documented, enhancing traffic flow, reducing congestion, and improving road safety. For instance, a study in (10) trains a ConvNet with annotated images to identify vehicle locations and uses a Support Vector Machine (SVM) for classification. Similarly, (11) proposes a vehicle type classification system using Faster R-CNN, demonstrating time efficiency and strong performance.

The work in (12) replaces traditional methods with DeepSort for car detection using a large dataset, showing that YOLOv4 outperforms YOLOv3. In (13) , YOLOv8 is used for real-time object detection, outperforming YOLOv5.

In (3), a two-stage deep learning approach for vehicle detection and classification is proposed, addressing challenges like diverse viewing angles and lighting conditions. The study in (1) compares R-CNN-based methods with YOLO versions, noting the trade-offs between speed and performance, highlighting YOLOv5's real-time capabilities.

(16) reviews vehicle detection methods, contrasting traditional and deep learning approaches, and highlights the effectiveness of CNNs and Faster R-CNN for vehicle detection and classification (23) . introduces IWDADL-VDC, combining YOLOv7 and DLSTM for remote sensing vehicle detection with hyperparameter tuning, showing improved performance.

Finally, (24) proposes a system for classifying vehicles on highways using various techniques to handle challenges like lighting changes and camera shake, achieving strong results.

The deep convolutional network (CNN) uses a two-stage method (25) for candidate box creation and classification, while the YOLO framework (6) employs a one-stage method, converting bounding box positioning into a regression problem. Over time, YOLO has evolved, with version 4 significantly improving object detection speed and accuracy.

3 YOLO Algorithm

The world of computer vision is constantly evolving, and the YOLO (You Only Look Once) algorithm stands as a prime example of this progress. This groundbreaking approach leverages the power of Convolutional Neural Networks (CNNs) to achieve real-time object detection, a significant leap forward compared to traditional methods. This advancement is

crucial because current computer vision systems often struggle with the speed and accuracy needed to mimic human-like visual intelligence. YOLO tackles this challenge head-on, paving the way for automating tasks and achieving a new level of efficiency in various fields (6).

YOLO's journey has been one of continuous refinement. The first iteration, YOLOv1, prioritized a compact model size and rapid processing speed, making it ideal for real-time video detection. However, it wasn't without limitations. Subsequent versions addressed these issues and incorporated new techniques. YOLOv2 improved positioning accuracy, a crucial aspect for tasks like object tracking or manipulation by robots. YOLOv3 addressed a key challenge in object detection – the ability to find objects of varying sizes within an image. It introduced multi-scale features, allowing the model to effectively detect both large and small objects within the same frame. The evolution continued with YOLOv4 exploring optimization possibilities for different hardware setups. This made YOLO more versatile and accessible for a wider range of applications. YOLOv5 offered flexibility in model size, allowing users to choose between a compact model for mobile devices or a larger model for higher accuracy on powerful machines. It also introduced a user-friendly framework for training, making YOLO more accessible to developers. YOLOv6 and YOLOv7 continued pushing the boundaries, achieving superior performance and speed compared to previous versions. This made them suitable for even more demanding tasks. Finally, YOLOv8 prioritizes accuracy, making it a valuable tool for tasks like medical imaging where precise detection of objects is critical (28).

The architecture of YOLOv1 comprises 24 convolutional layers succeeded by 2 fully connected layers, integrating leaky rectified linear unit activations, except for the final layer which employs a linear activation function. Drawing inspiration from Google Net and Network in Network, YOLO utilizes 1×1 convolutional layers for channel reduction. The output consists of a fully connected layer that produces a grid of 7×7 with 30 values assigned to each grid cell, accommodating ten bounding box coordinates (2 boxes) along with 20 categories (34) . YOLOv2, also known as YOLO9000, introduces the Darknet-19 backbone with 19 convolutional layers and 5 max-pooling layers. Similar to YOLOv1, it utilizes 1×1 convolutions for parameter reduction and incorporates batch normalization for regularization. The YOLOv3 architecture replaces max-pooling layers with strided convolutions and incorporates residual connections. It consists of 53 convolutional layers, each with batch normalization and Leaky ReLU activation. Additionally, YOLOv3 uses 1×1 convolutions with residual connections across the entire network (35) . YOLOv4 introduces various modules, including CMB, CBL, UP, SPP, and PANet, for improved performance. It offers scaled-down (YOLOv4-tiny) and scaled-up (YOLOv4-large) versions, optimized for different hardware. YOLOv5 adopts a modified CSPDarknet53 backbone and includes augmentations such as Mosaic, MixUp, and others. It offers five scaled versions (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) to suit specific applications and hardware requirements (36) . YOLOv6 introduces a new backbone with RepVGG blocks, similar to YOLOv5 in terms of spatial pyramid pooling fast (SPPF) and Conv modules, but with a decoupled head for improved performance. YOLOv7 introduces the Extended efficient layer aggregation network (E-ELAN) and a model scaling strategy for concatenation-based models. It utilizes planned re-parameterized convolution and includes a bag of freebies for enhanced performance. YOLOv8 features a modified CSPDarknet53 backbone with a C2f module, incorporating a spatial pyramid pooling fast (SPPF) layer for accelerated computation. The head is decoupled to independently process objectness, classification, and regression tasks. The architecture of YOLOv8 represents a

progression from its predecessors, featuring a convolutional neural network divided into two primary components: the backbone and the head. The backbone is constructed on a modified version of the CSPDarknet53 architecture, incorporating 53 convolutional layers with enhanced cross-stage partial connections. Complementing this, the head encompasses multiple convolutional layers followed by fully connected layers, tasked with predicting bounding boxes, objectness scores, and class probabilities. Notably, YOLOv8 introduces a self-attention mechanism in the head of the network and integrates a feature pyramid network for multi-scaled object detection. This design enables YOLOv8 to selectively focus on different regions of an image and effectively detect objects of varying sizes and scales (37) . Table I gives a brief overview of evolution of YOLO.

Version	**Focus**	**Applications**	**Year**	**Architecture**
YOLOv1	Speed	Real-time video detection	2016	GoogLeNet-inspired
YOLOv2	Accuracy (Positioning)	Tracking/manipulation	2017	Darknet-19, VGG
YOLOv3	Multi-scale Object Detection	General object detection	2018	Darknet-53
YOLOv4	Hardware Optimization	Wider range of applications	2020	CSPDarknet53
YOLOv5	Flexibility (Model Size)	Powerful machines	2020	CSPDarknet53
YOLOv6	Performance	Demanding tasks	2021	EfficientRep, CSPDarknet53
YOLOv7	Performance	Demanding tasks	2022	ELAN, CSPDarknet53
YOLOv8	Accuracy	Medical imaging	2023	CSPDarknet53

Table 1 Key features and architectural evolution of YOLO.

The continuous development of YOLO highlights its increasing accuracy and versatility. This powerful algorithm is well-positioned to revolutionize various computer vision tasks, bringing us closer to achieving human-like visual intelligence in machines (14) . YOLO algorithm revolutionizes object detection through its innovative grid-based approach. Initially, the input image undergoes partitioning into a grid with dimensions $S \times S$. Each grid cell becomes the focal point for predicting a set of bounding boxes, ensuring a systematic examination of the entire image for potential object presence. This grid-based methodology lays the foundation for YOLO's efficiency and real-time capabilities (15). For each grid cell, YOLO predicts B bounding boxes, each characterized by critical parameters. The confidence score (Pc) reflects the algorithm's certainty regarding the presence of an object within the bounding box. Additionally, the coordinates of the box center (bx, by) and its dimensions (bh, bw) contribute to the comprehensive set of predictions. The output of YOLO takes the form of a tensor with dimensions $S \times S \times (B \times 5 + C)$, encapsulating confidence scores and bounding box parameters across different classes. To refine its predictions and avoid redundancy, YOLO incorporates Non-Maximum Suppression (NMS) as a post-processing step. NMS selectively retains the most confident bounding boxes while suppressing those with significant overlap, resulting in a concise and accurate prediction set. This mechanism enhances the algorithm's precision in identifying and localizing objects within an image. The confidence score calculation is a pivotal step in YOLO's workflow. Pc is determined by multiplying the probability (p) with the Intersection over Union (IOU) between the predicted and actual bounding boxes. This intricate calculation ensures that the confidence score not only considers the likelihood of object presence but also factors in the accuracy of the bounding box. The utilization of IOU contributes to the algorithm's reliability in assessing overlaps between predicted and ground truth bounding boxes. YOLO operates on a regression-based algorithm, estimating probabilities and bounding boxes for all images.

This regression approach allows the model to learn complex relationships between input features and output predictions, accommodating diverse object classes. Moreover, the distribution and placement of objects within the grid cells are carefully managed, with each cell associated with values indicating object presence, bounding box parameters, and category information. The amalgamation of YOLO's grid-based methodology, bounding box predictions, NMS, confidence score calculation, regression-based approach, and grid distribution techniques collectively empowers the algorithm with efficiency and accuracy in real-time object detection tasks. This holistic approach has made YOLO a widely adopted solution across various domains requiring rapid and precise object detection capabilities (38) .

Here we discuss Performance parameters of YOLO algorithm in order to make better understanding of it's evaluation.

mAP (Mean Average Precision) : mAP serves as a prevalent metric for assessing the effectiveness of object detection algorithms. It amalgamates precision and recall values across various Intersection over Union (IoU) thresholds. In YOLO, the model undergoes evaluation across different IoU thresholds to gauge its performance under diverse overlapping criteria (39) .

Precision and Recall

Precision denotes the ratio of correctly predicted positive instances to the total predicted positives, whereas recall represents the ratio of correctly predicted positive instances to the total actual positives. YOLO leverages precision and recall to analyze the balance between accurately detecting objects and mitigating false positives.

Intersection over Union (IoU): IoU quantifies the overlap between the predicted bounding box and the ground truth bounding box. It is computed as the intersection area divided by the union area of the two boxes. YOLO employs IoU thresholds to ascertain whether a prediction qualifies as a true positive or false positive. Typically, this threshold is set to a value such as 0.5.

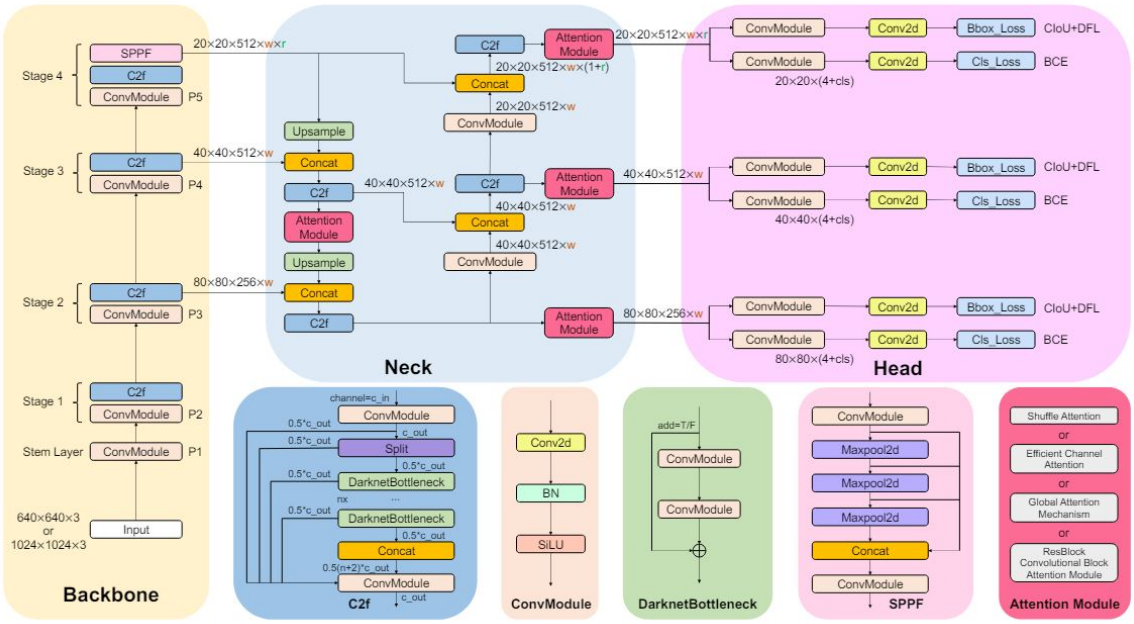
F1 Score: The F1 score serves as the harmonic mean of precision and recall, offering a balanced evaluation of a model's performance. It is utilized to assess the overall effectiveness of the model, taking into account both precision and recall.

Confidence Score: Confidence score represents the algorithm's confidence in its predictions. YOLO assigns confidence scores to each detected bounding box. The algorithm may consider bounding boxes with confidence scores above a certain threshold as positive detections

4 YOLOv8

YOLOv8, the latest chapter in the YOLO series of object detection models, stands out for its ability to achieve real-time performance while maintaining high accuracy and a lightweight structure. Building on the successes of its predecessors, YOLOv8 boasts a meticulously designed architecture with four key components: Backbone, Neck, Head, and Loss Function. The Backbone incorporates the Cross Stage Partial (CSP) (17) concept, offering the advantage of reducing computational loads while enhancing the learning capability of CNNs. YOLOv8 diverges from YOLOv5 employing the C3 module (18), adopting the

Figure 1: Detailed illustration of the YOLOv8 model architecture (31).



C2f module, which integrates the C3 module and the Extended ELAN (19) (E-ELAN) concept from YOLOv7 (20). Specifically, the C3 module involves three convolutional modules and multiple bottlenecks, whereas the C2f module consists of two convolutional modules concatenated with multiple bottlenecks. The convolutional module is structured as Convolution-Batch Normalization-SiLU (CBS). In the Neck part, YOLOv5 employs the Feature Pyramid Network (FPN) (22)architecture for top-down sampling, ensuring that the lower feature map incorporates richer feature information. Simultaneously, the Path Aggregation Network (PAN) (21) structure is applied for bottom-up sampling, enhancing the top feature map with more precise location information. The combination of these two structures is executed to guarantee the accurate prediction of images across varying dimensions.

YOLOv8 follows the FPN and PAN frameworks while deleting the convolution operation during the up-sampling stage, as illustrated in Figure 1. Finally, the Loss Function component plays a crucial role in optimizing the training process. YOLOv8 utilizes a composite loss function that incorporates several elements. The CIoU loss focuses on improving bounding box prediction accuracy. Additionally, there's a term for object classification loss, ensuring the model can correctly identify the detected objects. Lastly, a penalty term specifically addresses small objects within an image. This combination ensures not only accurate bounding box localization but also effective detection of even minor objects in a scene.

In conclusion, YOLOv8 builds upon the strengths of its predecessors, offering exceptional real-time object detection capabilities with high accuracy and a compact model size. Its meticulously designed architecture with efficient components like the C2f module and the streamlined Neck section empowers faster inference while maintaining precision.

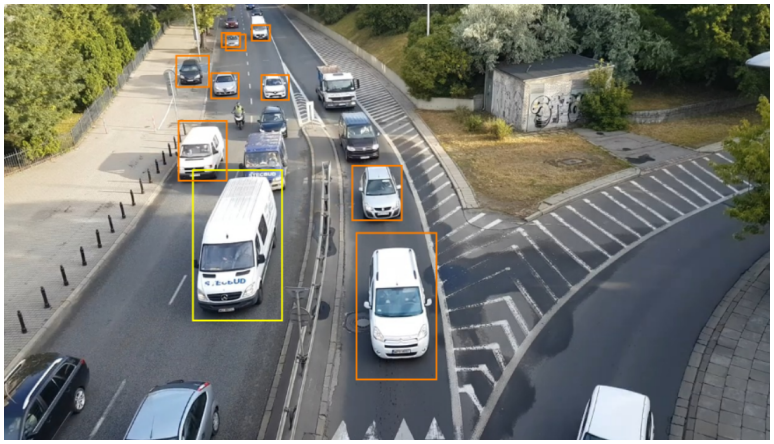
The focus on small object detection through the composite loss function further expands its applicability. These advancements make YOLOv8 a compelling choice for various computer vision tasks requiring speed, accuracy, and efficient resource utilization.

5 Data Collection

Data gathering is the foundation for building effective models. The quality and relevance of data directly impact the model's ability to learn and perform well. Open-source datasets often reflect the research needs of the time they were created, and their characteristics may not align perfectly with current research requirements (2). For our research, we utilized a combination of existing datasets and our own newly established datasets to verify our findings (33).

Since existing video datasets often lacked the necessary diversity, we ensured our collection captured footage at different times of the day, for instance fair days, low-light conditions, and even different type of vehicles such as iranian vehicles. By encompassing a wide range of environmental conditions, we aimed to create a more robust and versatile dataset that could train a detection model capable of performing reliably under various circumstances. As shown in Figures 2, 3 , 4 and 5 our dataset includes labeled images that reflect this diversity.

Figure 2: Classified labeled image 1 in the dataset labeled in roboflow (29)



After collecting the data, we painstakingly used Label Studio to meticulously set bounding boxes around objects of interest in each image and video frame. This manual labeling process was crucial for training the detection model with accurate data. The precision in labeling ensures that the model learns to identify and classify vehicles correctly, which is essential for applications in traffic management and safety.

Figure 3: Classified labeled image 2 in the dataset labeled in roboflow (29)

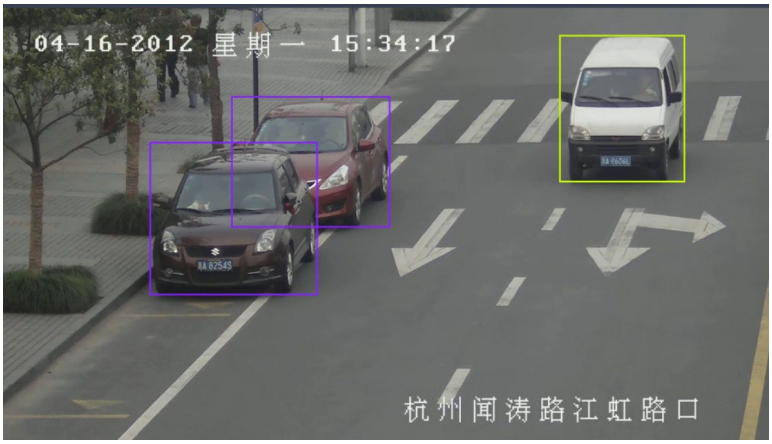
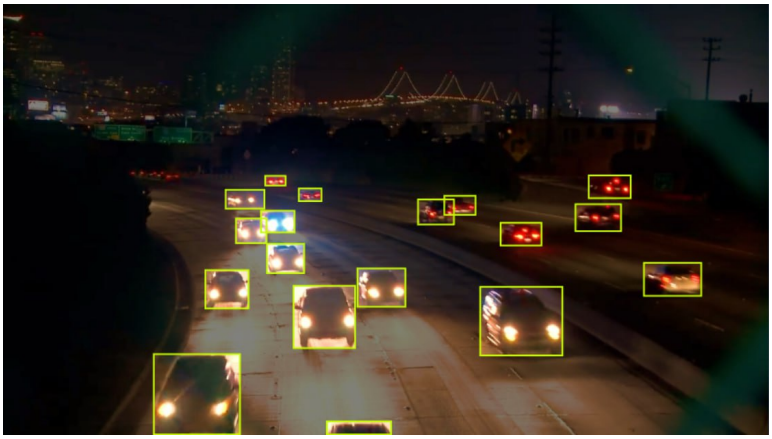


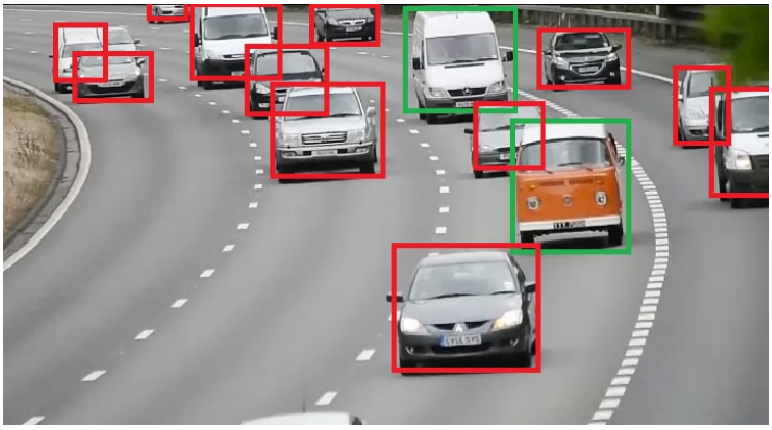
Figure 4: Classified labeled image 2 in the dataset labeled in roboflow (29)



6 Vehicle Classification

Within the collected dataset, we focused on classifying vehicles into four distinct categories: trucks, buses, cars and motors. This categorization aligns with the specific needs of expressway monitoring, where understanding the types of vehicles present is crucial for traffic management and safety applications. Heavy cars, such as trucks and buses, have different braking distances and require more space compared to light cars. Motors, including motorcycles and scooters, further complicate traffic flow due to their smaller size and maneuverability. By accurately classifying vehicles into these categories, the detection model can provide valuable insights for optimizing traffic flow and ensuring safety. To illustrate the diverse scenarios captured in the dataset, three representative images are presented in Figure 6. These images are only a few examples, and many more scenarios have been labeled to ensure comprehensive coverage of different conditions and vehicle types.

Figure 5: Classified labeled image 2 in the dataset labeled in roboflow (29)



To complete the production of the standard dataset, it was necessary to analyze the characteristics of the application scenario of this dataset to optimize and improve the vehicle detection model. Based on statistical labeling information, a plot analysis was performed on the dimensions of the labeled vehicle categories and bounding boxes. Figure 6a depicts a scenario where the camera is positioned above the road, providing a bird's-eye view of the traffic below. This perspective is particularly useful for monitoring overall traffic flow and detecting lane usage patterns. The elevated vantage point ensures a comprehensive view of the road, capturing all vehicle types effectively.

In contrast, Figure 6b shows a scenario where the camera is placed at the side of the road, capturing the side view of passing vehicles. This perspective allows for a detailed examination of individual vehicles, facilitating accurate classification of trucks, buses, cars, and motorcycles based on their profiles. The side view is especially beneficial for identifying vehicle shapes and sizes, which are critical for classification tasks.Figure 6c captures a scenario in snowy conditions, showcasing the model's robustness in adverse weather. Snow can significantly impact visibility. Including such challenging conditions in the dataset ensures that the model is trained to handle a variety of real-world environments, thereby enhancing its reliability and performance in practical applications.

These scenarios are only a few examples of the diverse conditions represented in the dataset. The variety of perspectives and environmental conditions included in the data collection process is crucial for developing a robust vehicle detection model. By training the model on this comprehensive dataset, it can better generalize to different real-world situations, ensuring more reliable and accurate vehicle classification in diverse traffic scenarios.

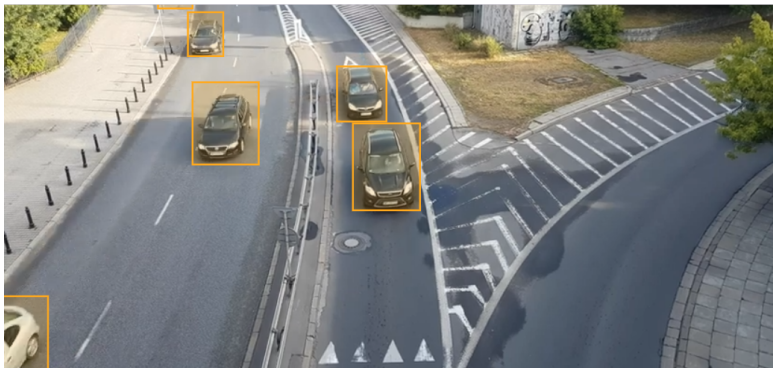
7 Implementation

7.1 Data Annotation

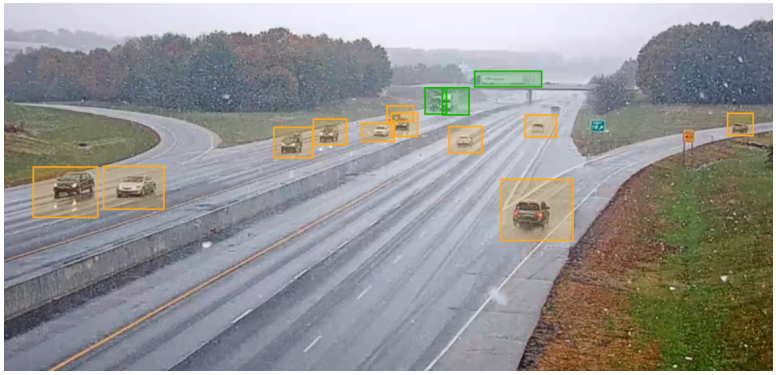
Building a powerful object detection model hinges on meticulously labeled training data. We employed advanced tools like Label Studio and Roboflow to achieve this crucial step. Our team drew tight bounding boxes around every desired vehicle within each image, ensuring

Figure 6: Different scenarios captured in the dataset via in roboflow (29)

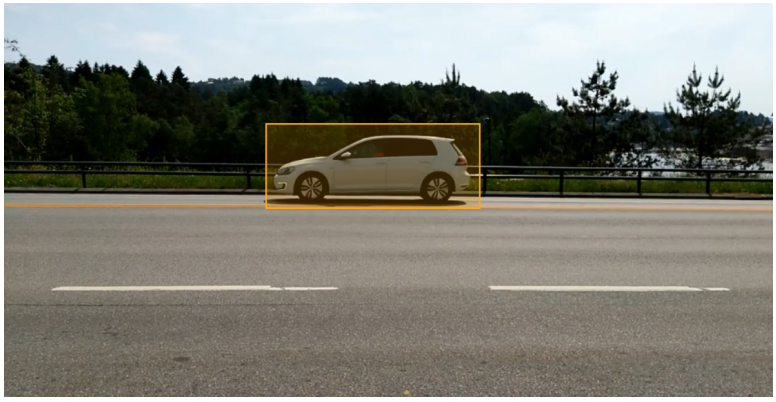
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



all their features were captured for the model's learning process. After this meticulous labeling, the data was uploaded to Roboflow for organized storage. Each image in this carefully curated dataset not only has bounding boxes but also detailed category information

for each object. This extra layer of data empowers the model to not only identify objects but also understand their specific categories. This focus on high-quality, meticulously labeled training data is the foundation for building accurate and effective object detection models.

The project began with the annotation of video frames using Roboflow, a powerful tool designed to facilitate the management and annotation of image data. Roboflow enabled us to collaborate effectively as a team by distributing tasks among members and ensuring consistent annotations across the dataset. The key features utilized in Roboflow included:

1. Collaborative Annotation: Each team member was assigned specific frames to annotate, ensuring the workload was evenly distributed.
2. Dataset Management: Roboflow automatically divided the annotated data into training, testing, and validation sets.
3. Export Options: After completing the annotations, we exported the dataset, which included 1500 images extracted from video frames, corresponding label files, and a YAML file for easy integration with machine learning frameworks.

8 Working With Label Studio

We started working on the Label Studio platform, attempting to label and prepare our dataset. However, the process was extremely time-consuming since we had to go frame by frame and manually apply all the labels. Given the time constraints, we began researching better platforms for this purpose, which led us to Roboflow. This platform accepts data (videos or images) and, after specifying the types of labels and the required tasks, it starts labeling automatically. We can then review the results image by image and manually correct any issues using the provided tools (32) .

Figure 7: Create a project in Label Studio for annotation (32)

8.1 Steps for Using Label Studio

To provide a detailed overview, here are the steps we followed while working with Label Studio:

Figure 8: Name and describe the project in Label Studio (32)

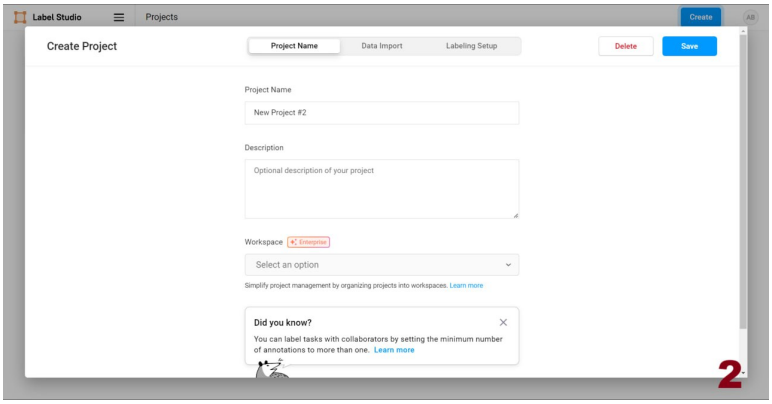


Figure 9: Upload the video or image file in Label Studio (32)

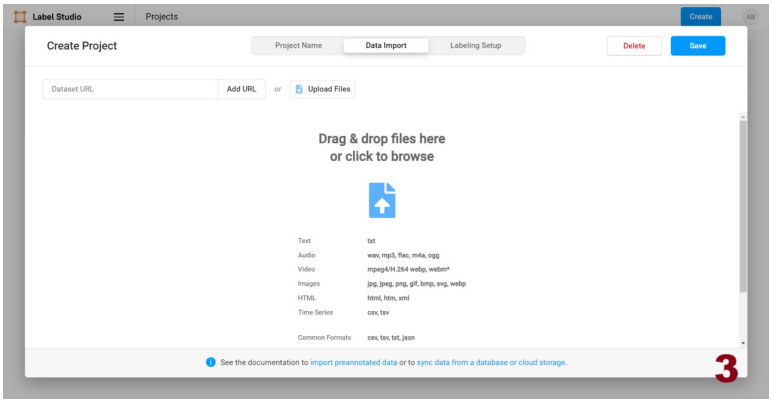


Figure 10: Specify the Type of computer vision task in Label Studio (32)

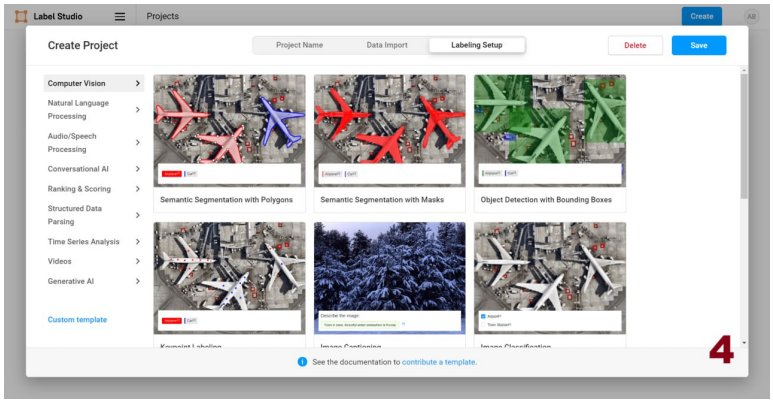
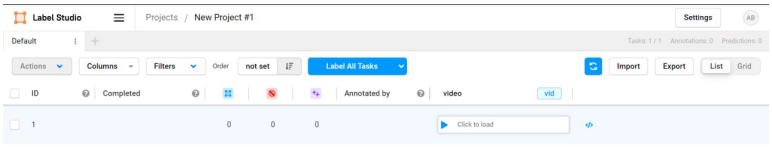
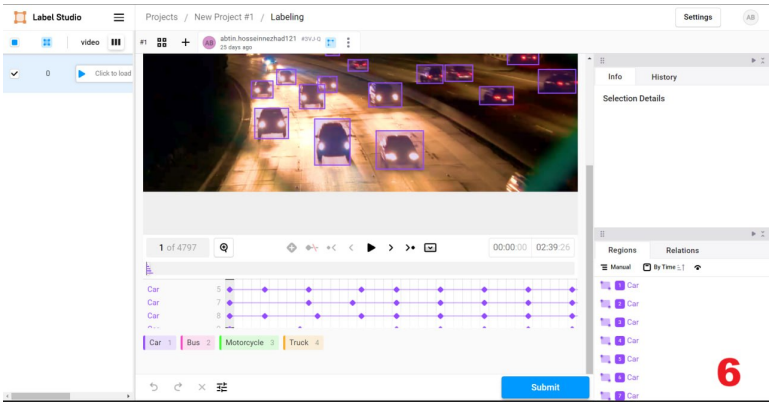


Figure 11: Enter the labeling environment in Label Studio (32)



5

Figure 12: Manually label frame by frame in Label Studio (32)



6

1. Create a Project: The first step in Label Studio is to create a new project. This can be done by navigating to the dashboard and clicking on the "Create Project" button (Figure 7).

2. Name and Describe the Project: After initiating the project creation, we provided a name and, if necessary, a description for the project. This helps in organizing and distinguishing between multiple projects, especially when managing a large dataset or numerous labeling tasks (Figure 8).

3. Upload the Video or Image File: Once the project was set up, we proceeded to upload our data. Label Studio supports various formats, so we could upload either video files or images. The platform allows batch uploads, which is helpful when dealing with multiple files (Figure 9).

4. Specify the Type of Computer Vision Task: After uploading the data, we specified the type of computer vision task we intended to perform. Label Studio supports a variety of tasks such as object detection, image classification, and segmentation. Choosing

the correct task type is crucial as it determines the labeling tools and functionalities available in the subsequent steps (Figure 10).

5. Enter the Labeling Environment: With the data uploaded and the task type defined, we entered the labeling environment. This is where the actual labeling takes place. Label Studio provides a user-friendly interface with various tools designed to facilitate the labeling process (Figure 11).

6. Manually Label Frame by Frame: In the labeling environment, we manually labeled the data frame by frame. This step involved painstakingly selecting and labeling objects in each frame, ensuring consistency across the dataset. The manual nature of this task made it particularly time-consuming, as labeling a 2-minute video could take up to two days to complete (Figure 12).

Despite the intuitive interface and comprehensive toolset provided by Label Studio, the manual labeling process proved to be highly labor-intensive and time-consuming. This experience prompted us to explore alternative platforms that could expedite the labeling process.

8.2 Transition to Roboflow

Given the challenges we faced with Label Studio, we began researching better platforms for dataset labeling, which led us to discover Roboflow. Roboflow offers a more automated approach to data labeling. By accepting data in the form of videos or images and allowing us to specify the types of labels and tasks required, Roboflow significantly reduces the manual workload. The platform automatically processes the data and generates labels, which we can then review and manually correct using the provided tools.

9 Working With Roboflow

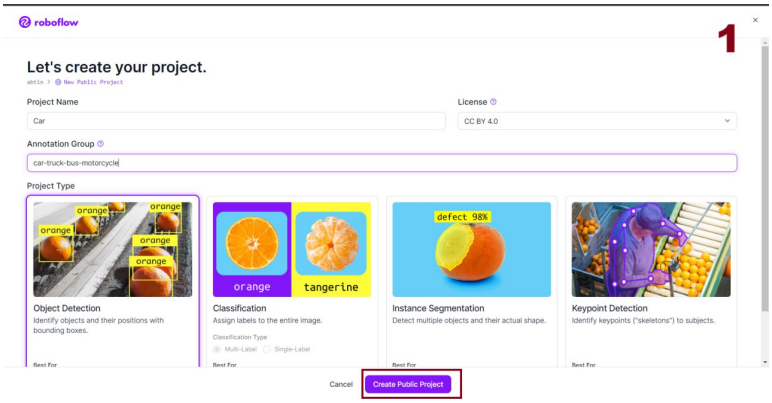
Roboflow is a comprehensive platform for managing, annotating, and training computer vision models. This guide provides a step-by-step process to create a project, upload data, and train a model using Roboflow (29).

9.1 Creating a Project

1. Navigate to the Roboflow Dashboard: Go to `https://app.roboflow.com` and log in or sign up for an account.

2. Create a New Project: Click on "New Project". Fill in the project name, select the appropriate license, and define the annotation group. For instance, in Figure 13, the annotation group includes ‘car‘, ‘truck‘, ‘bus‘, and ‘motorcycle‘.

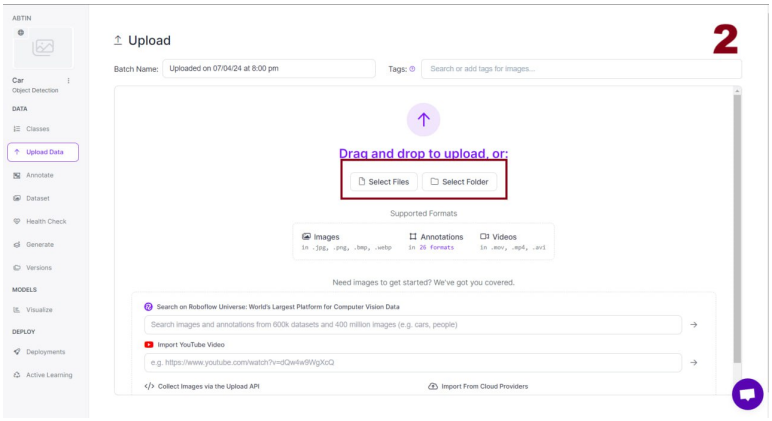
Figure 13: Creating a project in Roboflow for annotation (29)



9.2 *Uploading Data*

1. Upload Images: Click on "Upload Data". Drag and drop your image files into the upload area or select files from your computer as shown in Figure 14.
2. Organize and Tag: Optionally, add tags to organize your dataset for easier management.

Figure 14: Uploading data to Roboflow (29)



9.3 *Automated Annotation*

Roboflow Auto Label allows you to use large foundation vision models or Roboflow trained models to automatically label images (30) .
Employ Roboflow Auto Label for annotating commonplace items like vehicles, individuals, defects, and generic goods. However, it is not adept at discerning specific variations of an object (30) . The process will be demonstrated in the subsequent steps.

1. Upload Data: First, upload data to Roboflow as shown in (Figure 15).

2. Select Automated Labeling: Choose "Auto Label" after uploading your images(Figure 16) .

Figure 15: Drag and drop images in Roboflow (29)

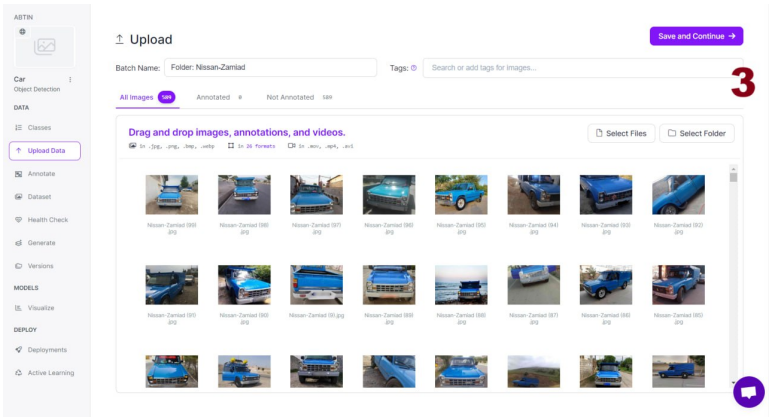
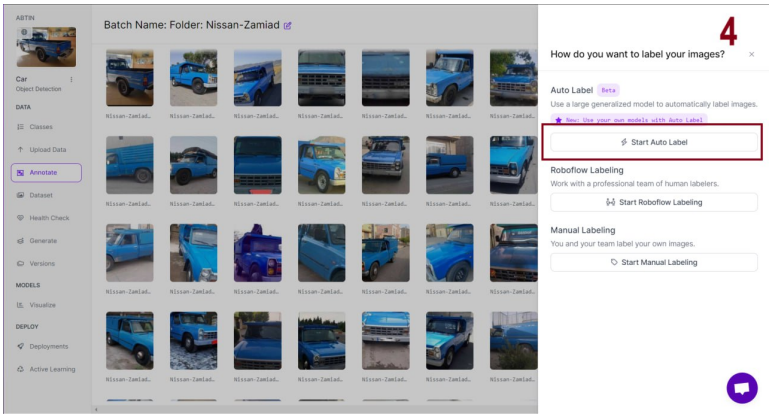


Figure 16: Start auto label (29)



9.4 Evaluate Roboflow Auto Label Labels

To evaluate Roboflow Auto Label Labels, follow these steps:

1. Understanding the Output: Review the annotations generated by Roboflow AutoDistill to understand which objects have been detected and labeled. See Figure 17 .

2. Comparison with Ground Truth: Compare the auto-generated labels with manually annotated data (ground truth) to identify any discrepancies.

3. Performance Metrics: Calculate performance metrics such as precision, recall, and F1-score to quantify the effectiveness of auto labeling.

4. Iterative Improvement: Use feedback from evaluation metrics to refine and improve the auto labeling process, adjusting parameters or retraining models if necessary.

Figure 17: Example of Roboflow AutoDistill output (29)

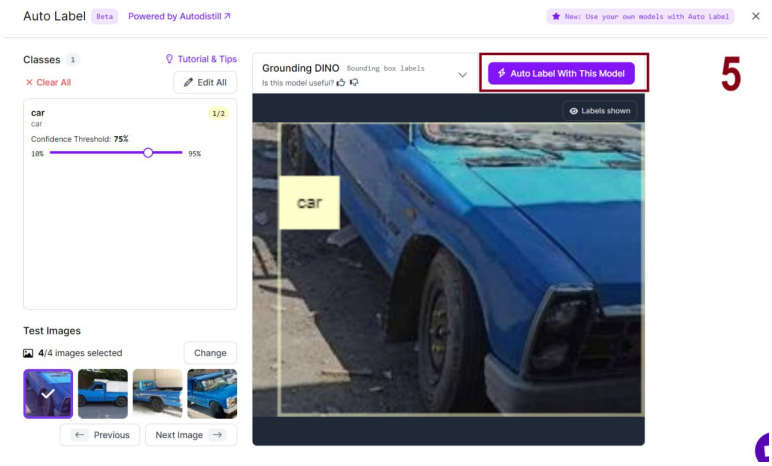


Figure 18: Annotating section in Roboflow (29)

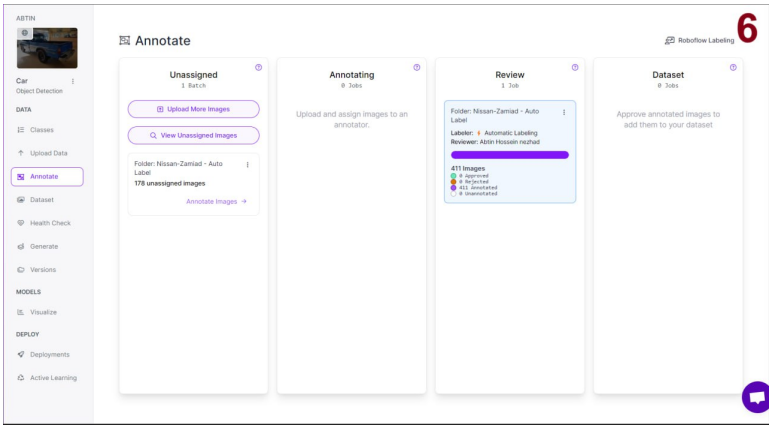
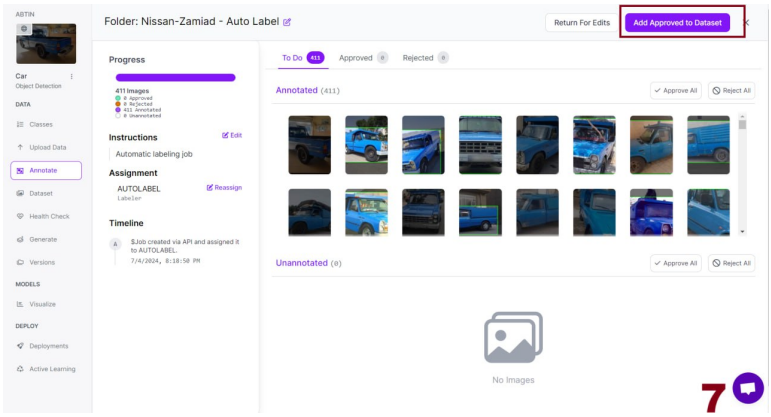


Figure 19: Add images to dataset in Roboflow (29)



9.5 Add Images to Dataset

Once you have annotated your image, you need to add it to your dataset for use in training a model (30) . Follow these steps:

1. Exit Annotation Tool: Exit out of the annotation tool.
2. Access Annotate Section: Click on "Annotate" in the sidebar.
3. Review Annotated Image: In the "Annotating" section, find the card indicating one annotated image for review (Figure 18) .
4. Add Image to Dataset: Click "Add Images to Dataset" to include your annotated image into the dataset (Figure 19) .

9.6 Labeling with Smart Polygon Tool

You can point and click anywhere on an image to create a label. When you hover over an object, a red mask will appear that lets you see what region of the object Smart Polygon will label if you click as shown in Figure 20 (30) .

9.7 Dataset after Adding Images

Updated Dataset View: The main interface shows the images divided into their respective categories: Train, Valid, and Test. Each image is tagged with its respective category (e.g., Train, Valid, Test) (30) . The images now appear with colored tags indicating their set assignments (Figure 21).

9.8 Review

You can individually approve or reject annotated images and send them back to the annotator for rework when necessary. To do so, click on a batch of images (30). Then, navigate between the Approved, Rejected, and To Do tabs to view the state of images in a batch (Figure 22).

Figure 20: Labeling with Smart Polygon in Roboflow (29)

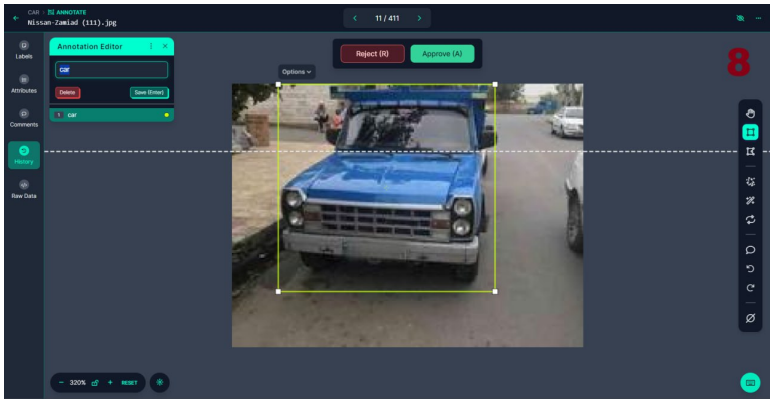
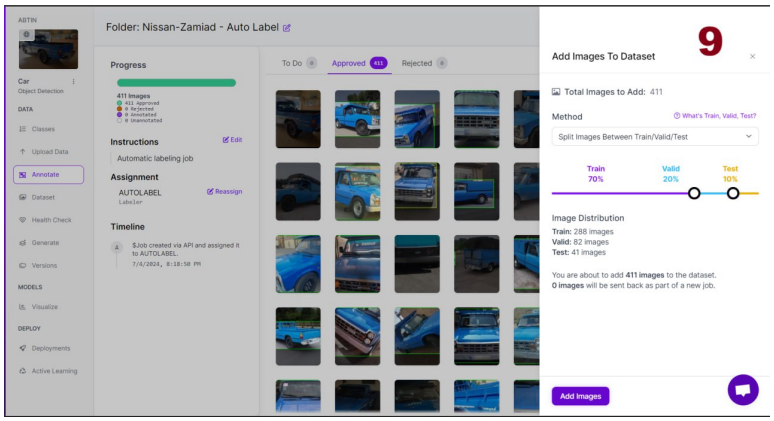


Figure 21: Adding images in Roboflow (29)



9.9 Additional Preprocessing

The image below shows the process of adding preprocessing steps in Roboflow. Preprocessing helps to standardize and improve the performance of your dataset by applying transformations to all images (30) . In this step, two preprocessing methods are highlighted:

1. Auto-Orient: Automatically adjusts the orientation of the images.

2. Resize: Resizes images to specified dimensions (640x640 pixels in this example).

You can add more preprocessing steps by clicking the "Add Preprocessing Step" button as shown in Figure 23.

Ensuring your dataset is in a standard format is crucial for effective machine learning model training. Preprocessing steps prepare your images for the model by standardizing various aspects (30). Unlike data augmentation, which is specific to the training set, preprocessing applies to all your data splits (Train, Validation, and Test) (Figure 24).

Figure 22: Review mode in Roboflow (29)

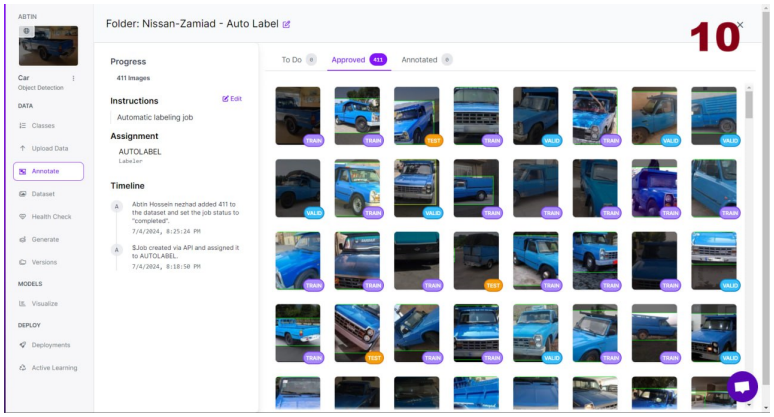


Figure 23: Adding preprocessing steps in Roboflow (29)

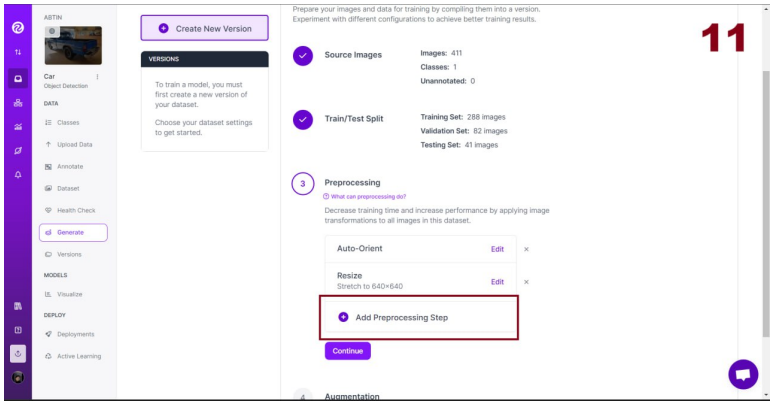
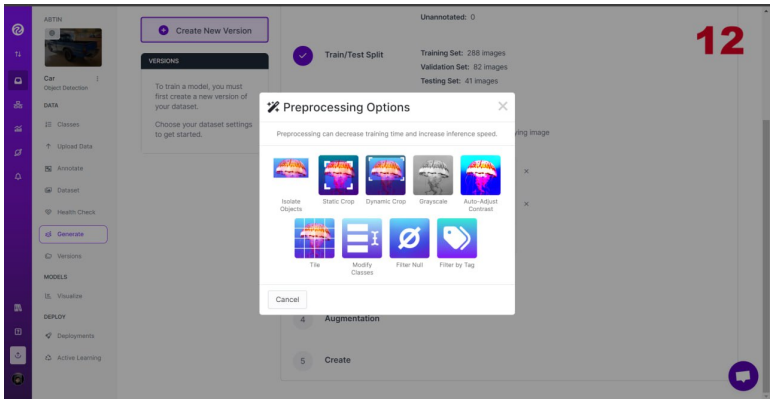


Figure 24: Preprocessing workflow options in Roboflow (29)



9.10 Export Dataset

You can export data from Roboflow at any time. You can export data using the Roboflow web interface or our Python package (30). To export data, first generate a dataset version in the Roboflow dashboard. You can do so on the "Versions" page associated with your project (Figure 24). After you have generated a dataset, click "Export" next to your dataset version (Figures 25 and 26) .

Figure 25: Export dataset in Roboflow (29)

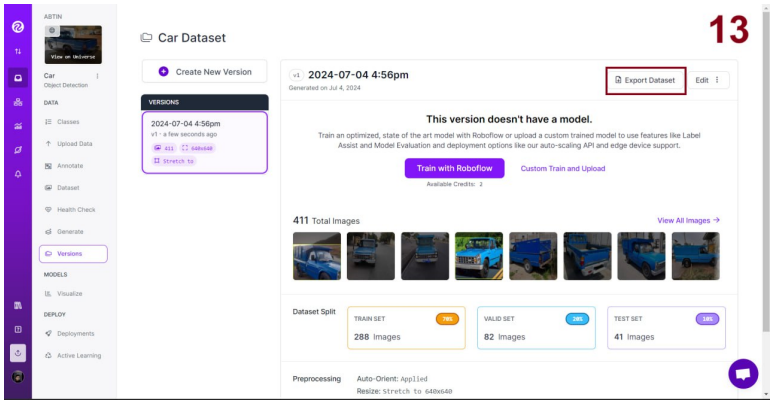
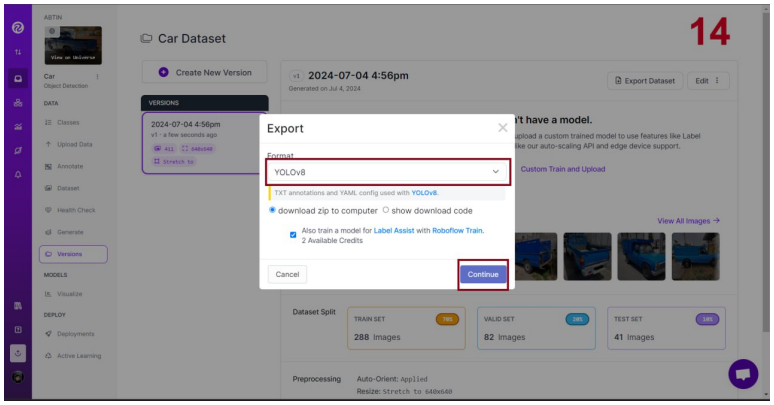


Figure 26: Choose version in Roboflow (29)

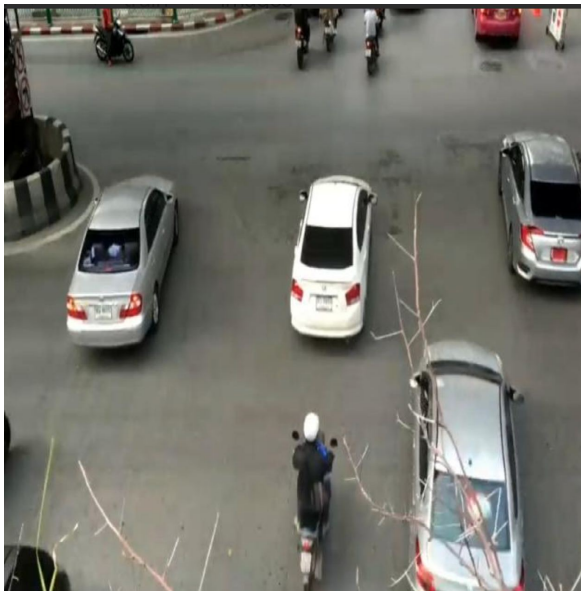


9.11 Data Preparation and Augmentation

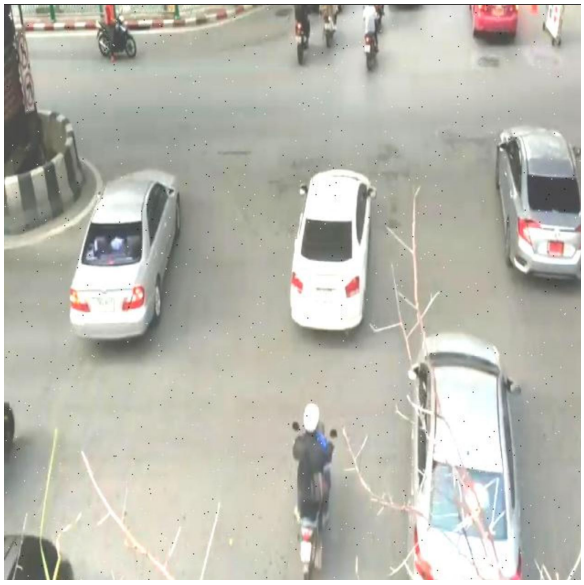
The exported dataset was uploaded to Google Drive for centralized access and further processing. To enhance the robustness of our model and prevent overfitting, we performed data augmentation on the training set. We applied various augmentation techniques to each image and saved the augmented images and their corresponding labels in the merged

Figure 27: Before and after augmentation technique to add noise to the tain set (29)

(a) Before



(b) After



directories. The augmentation techniques included rotation, scaling, flipping, and color jitter. The `augment_image` function applied various transformations to create diverse training examples, helping the model generalize better and perform well on unseen data. These augmentation techniques were critical in expanding the dataset and making the model

more resilient to variations in real-world data. By augmenting the training dataset and merging it with the original data, we created a comprehensive and diverse training set that enhanced the model's ability to recognize and classify vehicles accurately under different conditions. This step was crucial for building a robust model capable of performing well in varied scenarios (Figure 27) .

10 Model Training

10.1 Preparing the Environment: Installing Essential Libraries

Embarking on the journey of object detection with YOLOv8 begins with setting up the right environment. To harness the power of YOLOv8, we start by installing essential libraries. The ultralytics library provides the foundation for our object detection model. We also install a suite of other libraries numpy, pandas, matplotlib, scikit-learn, torch, torchvision, and torchmetrics ensuring we have all the tools needed for data manipulation, training, evaluation, and visualization.

10.2 Setting Up Data Access: Mounting Google Drive

Once our environment is prepared, we turn our attention to accessing our dataset. By importing necessary Python libraries and mounting Google Drive, we seamlessly integrate our data storage with the working environment. Google Drive serves as a repository for our dataset, allowing us to easily access training, validation, and test images stored in well-defined paths. These paths are essential for the next step, where we configure our dataset.

10.3 Configuring the Dataset: Creating a YAML File

Creating a YAML configuration file is a crucial step that defines our dataset for YOLOv8. This file specifies the paths to our training and validation images, the number of classes we are detecting, and their names. This setup ensures that our model knows exactly where to find the data and what it is looking for, paving the way for an efficient training process.

10.4 Verification and Readiness: Checking ultralytics Installation

With our dataset configured, we verify the installation of the `ultralytics` package. This step ensures that our environment is correctly set up and ready to train the YOLOv8 model. We then dive into the training phase, where we utilize the YOLOv8 small model (`yolov8s.pt`) and our configured data. We specify parameters such as the number of epochs, batch size, and the directory to save our results. This rigorous training process iteratively improves the model's ability to detect objects in our dataset.

10.5 Training the Model: Implementing YOLOv8

Training completes, and we transition to making predictions. Using the best model weights obtained during training, we apply the model to our test images, setting a confidence threshold to filter out less certain predictions. This step is where our model demonstrates its learned capabilities, identifying objects in new, unseen images.

10.6 Making Predictions and saving results

To preserve our results and facilitate further analysis, we copy the prediction results back to Google Drive (Figure 28) . This ensures that our work is saved and accessible for any subsequent steps, whether for evaluation or presentation.

10.7 Evaluation and Metrics: Assessing Model Performance

Evaluating the model is a vital step in understanding its performance. We import additional libraries and load the trained model to calculate evaluation metrics. This step provides insights into how well our model performs on the validation set, offering a quantitative measure of its accuracy and efficiency. Specifically, we calculate metrics such as recall 0.87 (Figure 33) , precision 1.00 at confidence level 0.873 (Figure 35), and f1-score 0.79 at 0.265 (Figure 32) .

These metrics help us understand how effectively the model identifies relevant objects (precision), how well it captures all relevant instances (recall), and how balanced it is between precision and recall (F-score). These quantitative measures are crucial for fine-tuning the model and optimizing its performance for real-world applications, additional informations are explained below.

11 Detailed Analysis

11.1 Confusion Matrix

The provided confusion matrix (Figure 30) visualizes the performance of a classification model across different categories. The matrix compares the actual labels (True labels) with the predicted labels by the model, giving insights into where the model is performing well and where it is making errors.

1. Categories (Labels):
   (a) Bus
   (b) Car
   (c) Motor
   (d) Truck
   (e) Background
2. Confusion Matrix Values:
   (a) The values in the matrix represent the count of predictions for each pair of actual and predicted labels.
   (b) Diagonal values represent correct predictions, where the predicted label matches the true label.
   (c) Off-diagonal values represent misclassifications, where the predicted label does not match the true label.

11.1.1 Confusion Matrix Detailed Analysis

1. Bus:
   (a) Correct Predictions: 8
   (b) Misclassifications:
      i. Predicted as Car: 5
      ii. Predicted as Motor: 1
      iii. Predicted as Truck: 3
      iv. Predicted as Background: 5
2. Car:
   (a) Correct Predictions: 644
   (b) Misclassifications:
      i. Predicted as Bus: 1
      ii. Predicted as Motor: 1
      iii. Predicted as Truck: 2
      iv. Predicted as Background: 139
3. Motor:
   (a) Correct Predictions: 302
   (b) Misclassifications:
      i. Predicted as Bus: 0
      ii. Predicted as Car: 2
      iii. Predicted as Truck: 1
      iv. Predicted as Background: 33
4. Truck:
   (a) Correct Predictions: 104
   (b) Misclassifications:
      i. Predicted as Bus: 1
      ii. Predicted as Car: 2
      iii. Predicted as Motor: 8
      iv. Predicted as Background: 5
5. Background:
   (a) Correct Predictions: 0 (None)
   (b) Misclassifications:
      i. Predicted as Bus: 1
      ii. Predicted as Car: 108
      iii. Predicted as Motor: 8
      iv. Predicted as Truck: 4

11.1.2 Observations

1. The model performs exceptionally well in predicting the Car category, with 644 correct predictions out of 800 total.

2. The Motor category also shows a high accuracy with 302 correct predictions.

3. The Truck and Bus categories have relatively lower correct predictions (104 and 8 respectively) but still show reasonable performance.

4. Background is consistently misclassified into various categories, indicating a potential area for model improvement.

5. The confusion between Car and Background is significant, with many background instances being incorrectly predicted as cars (139 instances).

The observations highlight both strengths and areas for improvement in the model's performance. The high accuracy in predicting cars and motorcycles suggests that the model has been effectively trained on these categories, likely due to sufficient representation and diversity of these vehicles in the dataset. This diversity allows the model to learn robust features and variations specific to cars and motorcycles, leading to accurate predictions.

Conversely, the lower accuracy in predicting trucks and buses indicates potential challenges in the model's ability to generalize to larger and less frequently encountered vehicle types. This could be attributed to insufficient diversity and representation of trucks and buses in the training data. Models trained on datasets lacking such diversity may struggle to distinguish these vehicles from others, impacting their overall performance in real-world scenarios where these larger vehicles are prevalent.

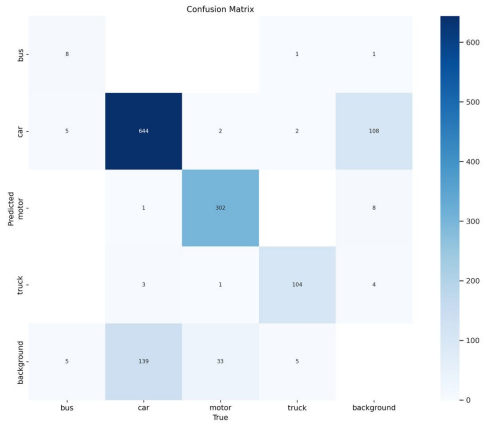
Moreover, the misclassification of background elements into various vehicle categories underscores the importance of data quality. A dataset with high-quality annotations and accurate labeling ensures that the model learns to differentiate between vehicles and background elements effectively. Improving data quality through meticulous annotation processes reduces such misclassifications, enhancing the model's precision and reliability.

The significant confusion between cars and background instances further emphasizes the need for diverse training data. By including a wide range of background scenarios and variations in lighting, occlusion, and environmental conditions, the model can better learn to differentiate between actual vehicles and background noise. This diversity helps in building a more robust model capable of accurately identifying vehicles in complex real-world settings.

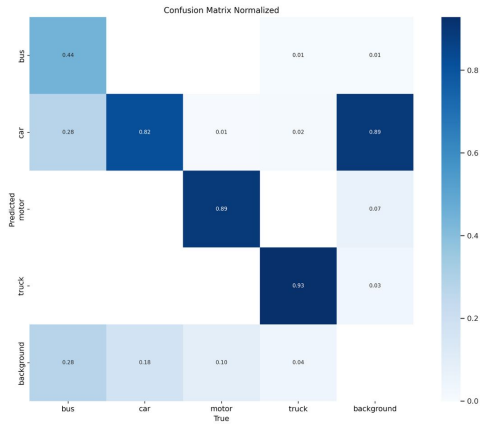
In conclusion, the diversity and quality of collected data directly influence the performance and robustness of a vehicle detection model. A diverse dataset that includes various vehicle types and environmental conditions ensures comprehensive training, enabling the model to generalize well and perform reliably across different scenarios. Quality data, with accurate annotations and representative samples, mitigates misclassifications and improves overall detection accuracy, making the model more effective for practical applications in traffic management and safety.

Figure 30: Confusion matrix of fine-tuned model

(a) Raw confusion matrix



(b) Normalized confusion matrix



11.2 Results

The figure above (Figure 31) visually represents the results discussed in this section. The model training and evaluation results are displayed over 20 epochs. The plots demonstrate the training and validation losses decreasing while precision, recall, and mean average precision (mAP) metrics exhibit fluctuations but show overall improvement. The results indicate that the model is learning effectively with respective losses decreasing and performance metrics improving over the epochs.

11.2.1 Metrics

1. **train / box loss**: This is the training loss for the bounding box prediction. It measures how well the predicted bounding boxes match the ground-truth bounding boxes.

2. **train / cls loss**clear: This is the training loss for the classification task. It measures the error between predicted class probabilities and the actual class labels.

3. **dfl / lossclear** : This stands for "Distribution Focal Loss" used during training. It usually aids in refining the predicted bounding box distributions.

4. **clearmetrics / precision(B)clear**: Precision metric for the model during training. Precision measures the proportion of true positive detections out of all positive detections predicted by the model.

5. **clearmetrics / recall(B)clear**: Precision metric for the model during training. Precision measures the proportion of true positive detections out of all positive detections predicted by the model.

6. **clearval / box lossclear**: This is the validation loss for the bounding box prediction. It serves the same purpose as the train/box_loss but on the validation set.

7. **clearval / cls lossclear**: This is the validation loss for the classification task on the validation set.

8. **clearval / dfl lossclear**: The Distribution Focal Loss on the validation set.

9. **clearmetrics / mAP50(B)clear**: Mean Average Precision at IoU 0.50. It's a standard object detection metric that measures the average precision for detecting objects considering a threshold IoU (Intersection over Union) of 0.50.

10. **clearmetrics/mAP50-95(B)clear**: Mean Average Precision across multiple IoU thresholds (from 0.50 to 0.95). This metric provides a more comprehensive evaluation of the model's detection performance.

11.3 Comprehensive Analysis

11.3.1 Train and Validation Loss Trends

- train/box_loss and val/box_loss: Both these training and validation box losses show a consistent decreasing trend over the epochs, indicating that the model is improving its ability to predict bounding boxes accurately.

- train/cls_loss and val/cls_loss: Similar to the box losses, these losses also exhibit a downward trend, suggesting that the model's classification accuracy is improving.

- train/dfl_loss and val/dfl_loss: The distribution focal losses are decreasing as well, implying better confidence in the predictions.

11.3.2 Precision and Recall

- metrics/precision(B): Precision shows fluctuations over time, but there's generally an improving trend. The fluctuations could be due to the model's varying performance on different batches of data.

- metrics/recall(B): Recall metric is steadily increasing, suggesting that the model is becoming better at identifying actual positives over the epochs.

11.3.3 Mean Average Precision (mAP)

- metrics/mAP50(B): This metric also shows fluctuations like precision but indicates improvement on average over the epochs.
- metrics/mAP50-95(B): This metric demonstrates a clear upward trend, implying that the model is getting better at detecting objects across various IoU thresholds.

11.3.4 Overal Analysis

The decreasing trends in losses (both training and validation) indicate that the model is learning and improving its performance. The improvement in recall and mAP metrics signifies an enhanced ability in detecting and classifying objects correctly. Although precision shows fluctuations, the general upward trend in most metrics (especially recall and mAP) paints a positive picture of the model's performance over the 20 epochs. The gradual decline in validation losses also suggests that the model is not overfitting and generalizes well to unseen data. The model appears to be on the right path, though more epochs might be needed for stabilization, especially for the precision metric.

11.4 Performance Metrics

11.4.1 F1 Score

The model achieves its highest F1 score of 0.79 when the confidence level threshold is set to 0.265. Adjusting the confidence threshold can impact the precision (Figure 34) and recall (Figures 33 and 35) and therefore the F1 score (Figure 32) . F1-Score is defines as below:

$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- **TP (True Positive):** The number of correctly predicted positive instances (or observations) by the model. In other words, TP is the count of instances that are actually positive and predicted by the model as positive.
- **TN (True Negative):** The number of correctly predicted negative instances (or observations) by the model. TN is the count of instances that are actually negative and predicted by the model as negative.
- **FP (False Positive):** Also known as Type I error, FP is the number of instances that are actually negative but predicted by the model as positive. In other words, FP is the count of instances that are falsely predicted as positive.
- **FN (False Negative):** Also known as Type II error, FN is the number of instances that are actually positive but predicted by the model as negative. FN is the count of instances that are falsely predicted as negative.

11.4.2 Recall

Recall measures the ability of the model to identify all relevant instances. It is the ratio of true positive predictions to the actual number of positives (True Positives / (True Positives

+ False Negatives)). The note accompanying the 'all classes' line (dark blue) indicates an overall accuracy (or recall rate at a certain confidence level) of 0.87 at a confidence threshold of 0.000, suggesting that the model performs quite well in terms of recall when no confidence threshold is applied. This chart demonstrates the trade-offs between model confidence and recall for different vehicle classes. It indicates that the truck class has robust performance, whereas the motor class is more volatile. The thick dark blue line provides a useful overview of the model's overall performance. Improving the consistency and maintaining high recall for all classes as confidence increases would be beneficial for enhancing the model's effectiveness (Figures 33 and 35).

$$\text{Recall} = \frac{TP}{TP + FN}$$

11.4.3 Prcision

The Precision-Confidence Curve chart illustrates the model's precision across various confidence levels for different vehicle classes. Precision, shown on the Y-axis, measures the proportion of true positive predictions among all positive predictions made by the model, reflecting the accuracy of positive predictions. Confidence, displayed on the X-axis, indicates the model's certainty in its predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

11.4.4 Precision-Recall Curve

The Precision-Recall Curve chart visualizes the trade-off between precision and recall for different vehicle classes. Precision (Y-axis) measures the accuracy of positive predictions, while Recall (X-axis) measures the ability to identify all relevant instances. This chart is particularly useful when evaluating models on imbalanced datasets.

1. Bus (light blue line): The bus class demonstrates a relatively poor performance with a rapid drop in precision as recall increases. This is reflected in its low average precision score of 0.434, indicating both precision and recall are challenging for the model in this category.
2. Car (orange line): The car class maintains high precision (>0.8) for a significant range of recall values before it starts to decline as recall approaches 1. This is confirmed by the high average precision score of 0.885.
3. Motor (green line): The motor class shows the best performance with almost perfect precision across all recall values. The average precision of 0.959 reflects this strong performance.
4. Truck (red line): The truck class also performs very well, with high precision maintained across almost the entire recall range. The average precision is 0.914, highlighting its robustness.

The combined performance of all classes shows a good balance between precision and recall, with high precision maintained until recall reaches around 0.8, after which precision declines. The mean average precision (mAP) of 0.798 at a recall threshold of 0.5 indicates strong overall effectiveness of the model.

Figure 28: Fine-Tuned model prediction outputs

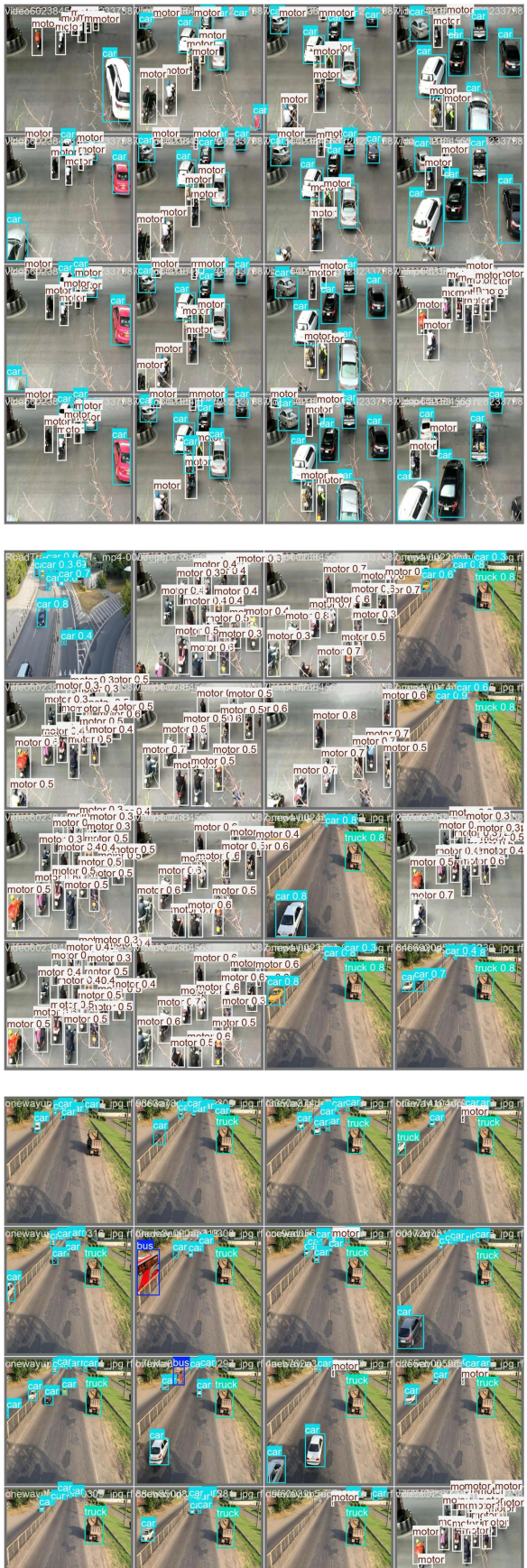


Figure 29: Sample outputs of fine-tuned yolo v8

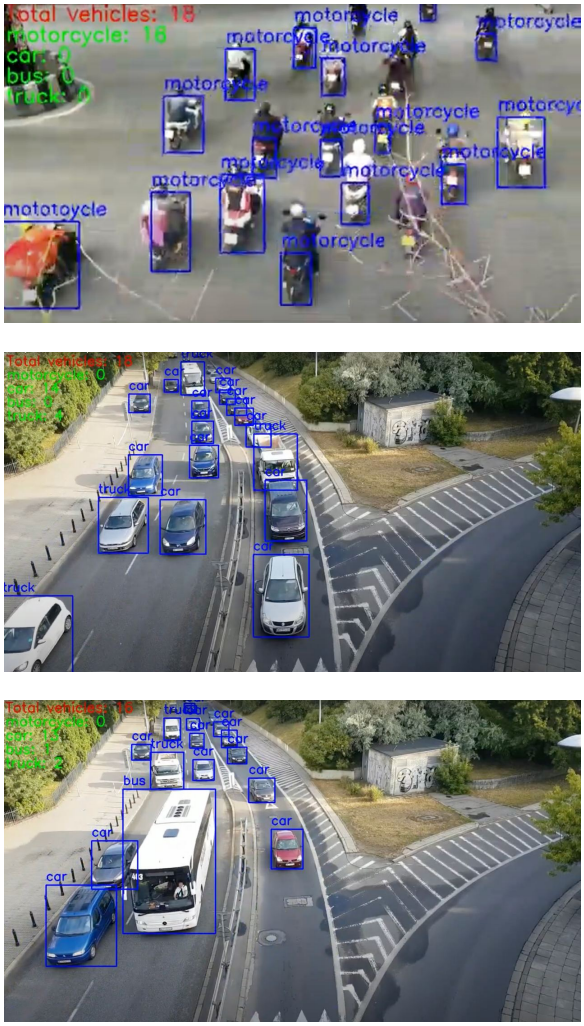


Figure 31: Result analysis through epochs

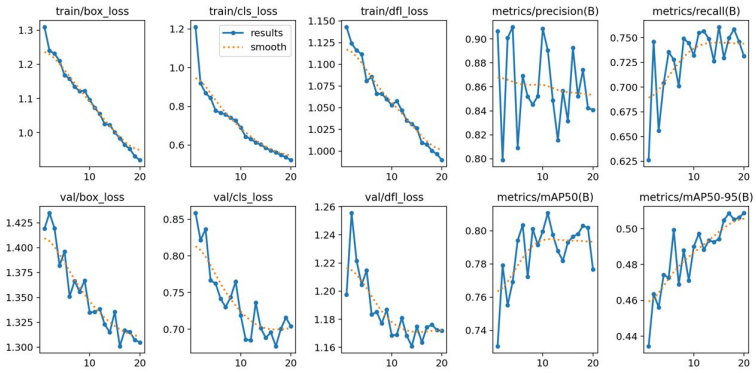


Figure 32: F1 score curve

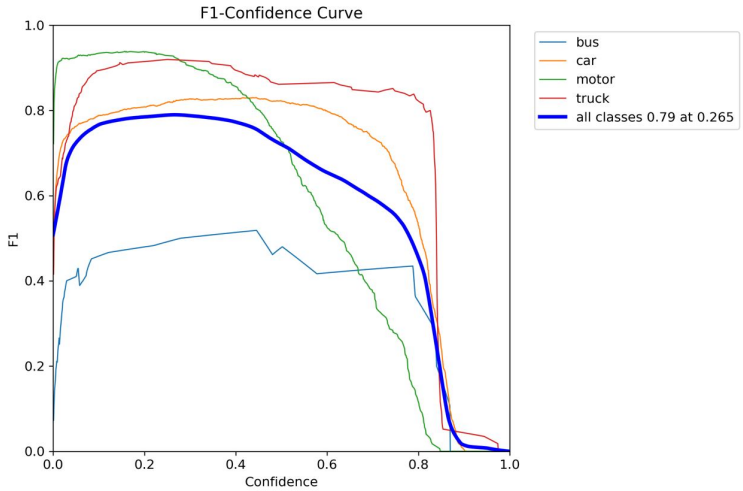


Figure 33: Recall curve

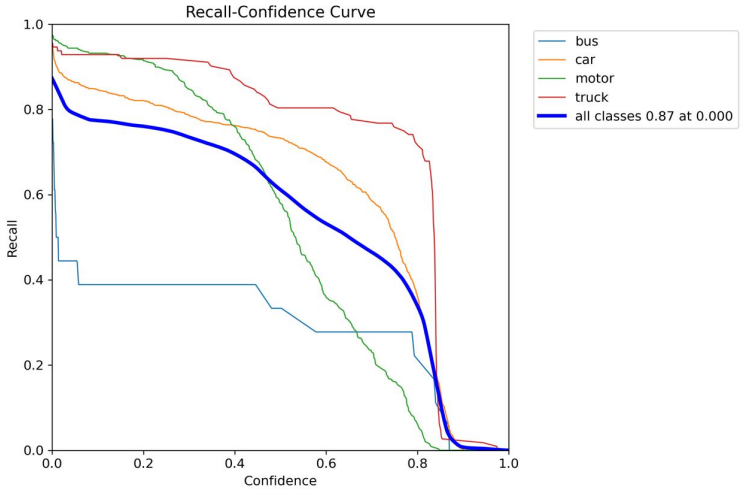


Figure 34: Precision curve

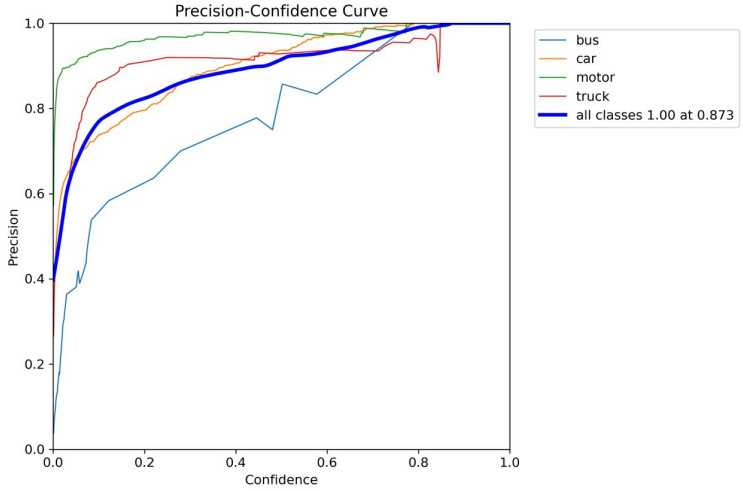
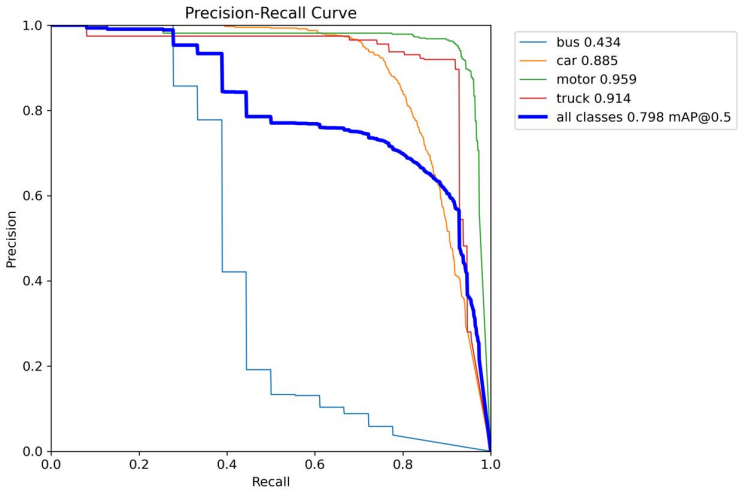


Figure 35: Precision-Recall curve



12 Conclusions

In this report, we have highlighted the effectiveness of the YOLOv8 deep learning algorithm for vehicle detection and classification within traffic monitoring systems. As urban environments become increasingly congested, the need for advanced traffic monitoring solutions has never been more critical. Our approach, which leverages the robust object detection capabilities of YOLOv8, offers significant improvements, making it highly suitable for practical applications in traffic analysis.

Our study involved meticulously training the YOLOv8 model using a custom dataset specifically curated for traffic scenarios. This dataset encompassed a variety of conditions, including diverse lighting and weather scenarios, ensuring the model's robustness and versatility. By taking a comprehensive approach to data collection and annotation, we enabled the YOLOv8 model to perform well in detection and classification across different vehicle categories, including heavy vehicles (buses and trucks) and light vehicles (cars and motorcycles).

The results of our research show that YOLOv8 meets the requirements for traffic monitoring systems. The model's ability to localize and classify vehicles under diverse conditions holds significant promise for its integration into intelligent traffic management systems. By providing detailed and precise vehicle classification, our system can enhance various applications, such as dynamic traffic light control, restricted area enforcement, and traffic law enforcement.

We also addressed inherent challenges in real-world traffic monitoring, such as variations in lighting, occlusion, and background clutter. By meticulously designing the dataset and employing advanced labeling techniques, we ensured that the YOLOv8 model could effectively handle these challenges, further validating its practical applicability.

In conclusion , our research demonstrates that the YOLOv8 deep learning algorithm presents a powerful solution for vehicle detection and classification in traffic monitoring systems. The integration of YOLOv8 into existing traffic monitoring infrastructures can significantly enhance traffic flow, reduce congestion, and improve road safety. Future research can explore further optimizations and applications of YOLOv8 in more complex traffic scenarios and other domains requiring object detection.

References

[1] M. Maity, S. Banerjee, S. S. Chaudhuri, "Faster R-CNN and YOLO based Vehicle detection: A Survey," in *Proceedings of the International Conference on Computational Mathematics and Computing*, 2021, doi: 10.1109/ICCMC51019.2021.9418274.

[2] Y. Zhang, Z. Guo, J. Wu, Y. Tian, H. Tang, X. Guo, "Real-Time Vehicle Detection Based on Improved YOLO v5," *Sustainability*, vol. 14, no. 19, 2022, doi: 10.3390/su141912274.

[3] A. M. Ghoreyshi, A. AkhavanPour, A. Bossaghzadeh, "Simultaneous Vehicle Detection and Classification Model based on Deep YOLO Networks," in *Proceedings of the International Conference on Machine Vision and Image Processing*, 2020, doi: 10.1109/MVIP49855.2020.9116922.

[4] Z. Gong, K. Guo, D. Yang, "Vehicle collaborative detection based on YOLOv5," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2024, doi: 10.54254/2755-2721/50/20241583.

[5] Do Thuan, "EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART OBJECT DETECTION ALGORITHM," Master's thesis, Oulu University of Applied Sciences, 2021.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv preprint arXiv:1506.02640*, 2016, doi: 10.1109/CVPR.2016.91.

[7] K. Shea, R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015, doi: 10.48550/arXiv.1511.08458.

[8] Z. Fan, Y. Zhu, Y. He et al., "Deep learning on monocular object pose detection and tracking: A comprehensive overview," *ACM Computing Surveys*, vol. 55, no. 4, 2022, doi: 10.48550/arXiv.2105.14291.

[9] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020, doi: 10.48550/arXiv.2004.10934.

[10] L. Yang, P. Luo, C. C. Loy, X. Tang, "A large-scale car dataset for fine-grained categorization and verification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2015, pp. 3973–3981.

[11] X. Wang, W. Zhang, X. Wu et al., "Real-time vehicle type classification with deep convolutional neural networks," *Journal of Real-Time Image Processing*, vol. 16, no. 1, 2019, pp. 5–14.

[12] M. A. B. Zuraimi, F. Hafizhelmi, K. Zaman, "Vehicle Detection and Tracking using YOLO and DeepSORT," *IEEE Symposium Series on Computational Intelligence*, 2021.

[13] D. Reis, J. Hong, J. Kupec, A. Daoudi, "Real-Time Flying Object Detection with YOLOv8," *IEEE Transactions on Image Processing*, vol. 28, no. 11, 2019, pp. 5682–5695.

[14] Juan Terven, Diana Margarita Córdova Esparza, Julio-Alejandro Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *IEEE Symposium Series on Computational Intelligence*, 2023.

[15] Eko Prasetyo, Nanik Suciati, Chastine Fatichah, "A Comparison of YOLO and Mask R-CNN for Segmenting Head and Tail of Fish," *IEEE Symposium Series on Computational Intelligence*, 2020.

[16] Shiyu Zhang, Lechao Wang, Bingbing Wang, Bo Yin, "A Review of Deep Learning for Vehicle Detection," *Artificial Intelligence Review*, vol. 53, no. 8, 2020, pp. 4837-4878.

[17] C.Y. Wang, H.Y.M. Liao, Y.H. Wu et al., "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 390–391.

[18] G. Jocher, K. Nishimura, T. Mineeva, R. Vilariño, "yolov5," *Code repository*, 2020.

[19] C.Y. Wang, H.Y.M. Liao, I.H. Yeh, "Designing Network Design Strategies through Gradient Path Analysis," *arXiv preprint arXiv:2211.04800*, 2022.

[20] C.Y. Wang, A. Bochkovskiy, H.Y.M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-The-Art for Real-Time Object Detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7464–7475.

[21] S. Liu, L. Qi, H. Qin, J. Shi, J. Jia, "Path Aggregation Network for Instance Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768.

[22] T.Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[23] V. Thavavel, M. Sivakumar, S. Shridevi, V. Subbiah Parvathy, J.V. Naga Ramesh, K. Syed, S.N. Mohanty, "Intelligent Water Drop Algorithm with Deep Learning-Driven Vehicle Detection and Classification," *AIMS Mathematics*, vol. 9, no. 5, 2024, pp. 11352–11371.

[24] "Author Not Available," "A Hybrid Algorithm for Vehicle Detection and Classification on Urban Highways," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[25] X. Hu et al., "SINet: A Scale-Insensitive Convolutional Neural Network for Fast Vehicle Detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 3, 2019, pp. 1010–1019, doi: 10.1109/TITS.2018.2838132.

[26] M. A. B. Zuraimi, F. Hafizhelmi Kamaru Zaman, "Vehicle Detection and Tracking using YOLO and DeepSORT," in *Proceedings of the IEEE Conference*, 2021, doi: 10.1109/978-1-6654-0338-2/21/$31.00 ©2021 IEEE.

[27] R. Girshick, J. Donahue, T. Darrell, J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587, doi: 10.1109/CVPR.2014.81.

[28] Bhavana Baburaj, Aaron J Roy, Brite Sun George, Therese Yamuna Mahesh, "A Comparative Study on Different YOLO Versions," *International Journal of Engineering Applied Sciences and Technology*, vol. 8, no. 12, 2024, pp. 219–222.

[29] Roboflow, Inc., "Getting Started with Roboflow," 2024, `https://blog.roboflow.com/getting-started-with-roboflow/`, Accessed: 2024-07-04.

[30] Roboflow, Inc., "Automated Annotation with Auto Label," 2024, `https://docs.roboflow.com/annotate/automated-annotation-with-autodistill`.

[31] Chun-Tse Chien, Rui-Yang Ju, Kuang-Yi Chou, Enkaer Xieerke, Jen-Shiun Chiang, "YOLOv8-AM: YOLOv8 with Attention Mechanisms for Pediatric Wrist Fracture Detection," 2024, Corresponding author: Jen-Shiun Chiang (jsken.chiang@gmail.com).

[32] LabelStudio, "Automated Annotation with Auto Label," 2024, `https://labelstud.io/guide/`.

[33] Pixabay, "Scooters, traffic, street, motorcycle," 2016, `https://pixabay.com/videos/scooters-traffic-street-motorcycle-5638/`, Accessed: 2024-07-12.

[34] Joseph Redmon, Santosh Divvala, Ross Girshick, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[35] Therese Yamuna Mahesh, Midhun P. Mathew, "Detection of Bacterial Spot Disease in Bell Pepper Plant Using YOLOv3," *Journal of Plant Pathology*, vol. 45, no. 3, 2023, pp. 123–130, doi: 10.1080/03772063.2023.2176367.

[36] Midhun P. Mathew, Therese Yamuna Mahesh, "Leaf-based Disease Detection in Bell Pepper Plant Using YOLO v5," *Journal of Plant Pathology*, vol. 43, no. 2, 2021, pp. 98–105.

[37] Muhammad Hussain, "YOLO-v1 to YOLO-v8: The Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," *Journal of Industrial Technology*, vol. 50, no. 4, 2023, pp. 200–210, doi: 10.1016/j.jindustrie.2023.03.005.

[38] Upulie Lakshini Kuganandamurthy, "Real-Time Object Detection using YOLO: A Review," *Journal of Computer Vision Research*, vol. 35, no. 1, 2021, pp. 45–52, doi: 10.13140/RG.2.2.24367.66723.

[39] Heng Li, Xufei Zhuang, Shi Bao, Junnan Chen, Chenxi Yang, "SCD-YOLO: a lightweight vehicle target detection method based on improved YOLOv5n," *Journal of Electronic Imaging*, vol. 33, no. 2, 2024, pp. 023041–023041, doi: 10.1117/1.JEI.33.2.023041.