UNIVERSITY OF BIRMINGHAM

Traffic Object Detection and Density Estimation Using YOLOv8

Erman Erbak
ID: 2545151

School of Computer Science
MSc Computer Science and Artificial Intelligence

Supervised By
Dr. Mian M. Hamayun

word count: 11858

Abstract

The developments in AI and ML have utterly changed the face of technologies that deal with image and video recognition, which, in turn brings huge benefits in fields like autonomous systems, security, smart transportation, etc. This dissertation presents the design and evaluation of a Traffic Object Detection and Density Calculation System using state-of-the-art latest developments in object detection algorithms. Central to this project is the YOLOv8 model, which is recognized for its rapidity and efficacy in real-time object detection applications. The primary aim of the system is to deliver a resilient and dependable approach for identifying traffic objects and assessing traffic density in real-time, utilizing datasets that have been specifically trained for this purpose.

Comparing object detection methods, such as Faster R-CNN, YOLOv5, and OpenCV with TensorFlow, makes YOLOv8 the most fit model for this system, offering more detection speed and precision. After training the YOLOv8n model from the ground up with our custom traffic dataset of 158 MB, outstanding system performance is evidenced by critical metrics such as mean Average Precision and Intersection over Union. The results indicate that the YOLOv8n(2023) model offers a remarkable balance between speed and detection precision, making it an optimal choice for real-time traffic monitoring applications. The system will not only identify the traffic entities effectively, but it also presents a novel methodology for estimating traffic density and hence qualifies as an effective, practical option for real-time traffic surveillance.

The results of this research encompass an optimized version of the YOLOv8n model, efficient real-time video processing frameworks, and a data storing system for the analysis of annotated traffic information. Future research will be focused on perfecting the detection algorithms, scalability of the system, and handling more complicated traffic scenarios.

Keywords

Real Time Object Detection, Traffic Object Recognition, YOLOv8, Machine Learning, Computer Vision, Video Processing, Traffic Density.

Acknowledgements

I would like to express my deepest appreciation to Dr Mian Mohammad Hamayun, who has been a very good teacher and mentor, and whose enthusiasm for “Computer Science and Artificial Intelligence” had lasting effects, for the supervision of my project and for his immeasurable guidance and support throughout the year.

I would also like to thank to all the faculty members of the University of Birmingham Computer Science Department for their help and encouragement, including Prof. Kashif Rajpoot who was my inspector during the dissertation process, and I am also truly grateful for my friends and family for believing in me, and for their continuous support.

I would also like to give speacial thanks to my life partner for her support, patience and encouragement.

1. Introduction

Object detection is one of the very basic tasks in computer vision. Hence, its application is very broad and ranges from traffic monitoring and autonomous driving to urban planning and intelligent transport systems. This skill of detecting, classifying, and estimating the identification and density of objects such as vehicles and people in real time is therefore very critical to modern smart city infrastructures. Object detection techniques have been undergoing rapid evolution during the last two decades. In fact, with the arrival of Deep Learning and CNNs-(Convolutional Neural Networks- (Zou et al., 2022), this evolution has happened at an unimaginable pace, thereby changing the face of this research field altogether. This evolution, from traditional handcrafted feature-based methods to modern CNN-based models like YOLO-or You Only Look Once-has gone on to realize much faster and much more accurate object detection systems that can deal with complex, real-world situations.

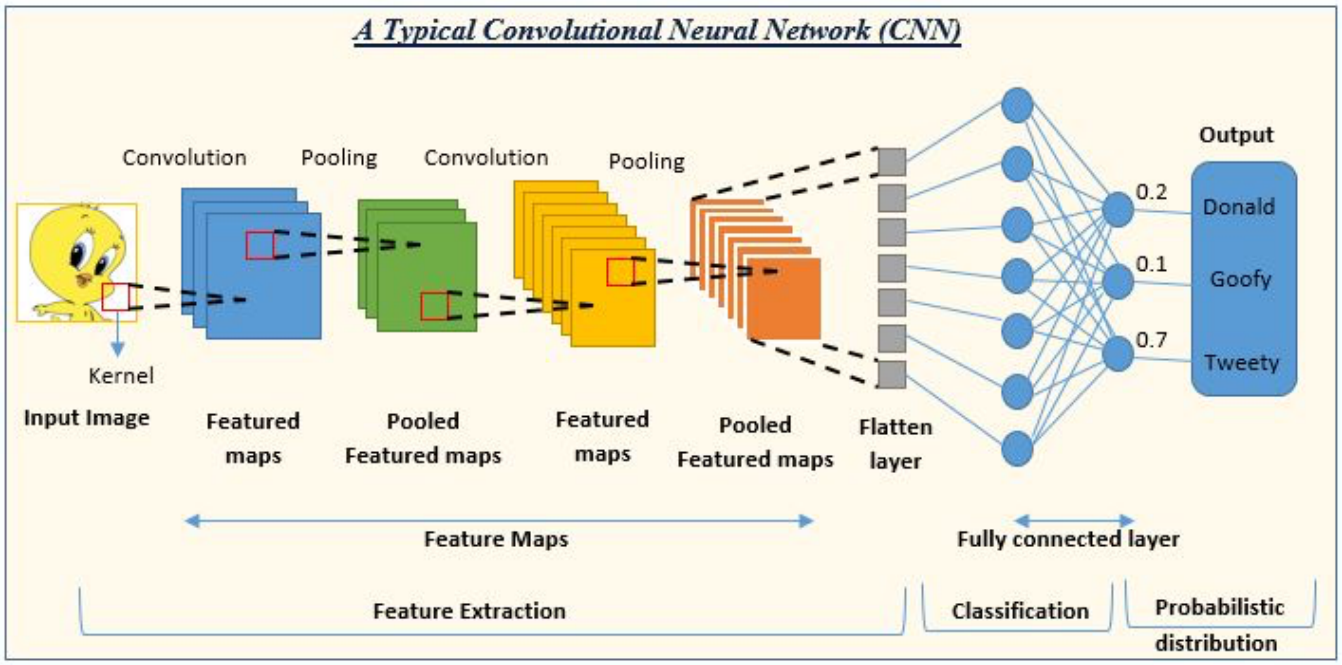


Figure -1: CNN, Convolutional Neural Network Layers

The Convolutional Neural Network (CNN) architecture, as illustrated in Figure-1, is fundamental in object detection due to its hierarchical feature extraction process. The convolutional layers apply learnable filters (kernels) to the input image, progressively capturing low-level features such as edges and textures, and higher-level features like shapes and objects. Pooling layers further reduce the spatial dimensions of the feature maps, enhancing computational efficiency and enabling scale-invariant detection. After flattening, the fully connected layers allow for classification by assigning probabilistic scores to different object classes, making CNNs ideal for object detection tasks where localization and classification occur simultaneously, as seen in advanced models like YOLO and Faster R-CNN.

Figure-2, illustrates the number of publications in object detection during the past two decades. It clearly shows the increasing interest and fast development in this area. There has been a steady increase in the output of research since the early 2000s, especially after 2014, to go hand-in-hand with the rise of deep learning. This can be very well accounted for by the introduction of CNNs and subsequent successful

usage in object detection tasks related to real-time vehicle detection for traffic systems. The figure acts as a testimony to the increase in the role of object detection technologies in a wide variety of sectors, right from autonomous vehicles to urban surveillance systems (Zou et al., 2022).

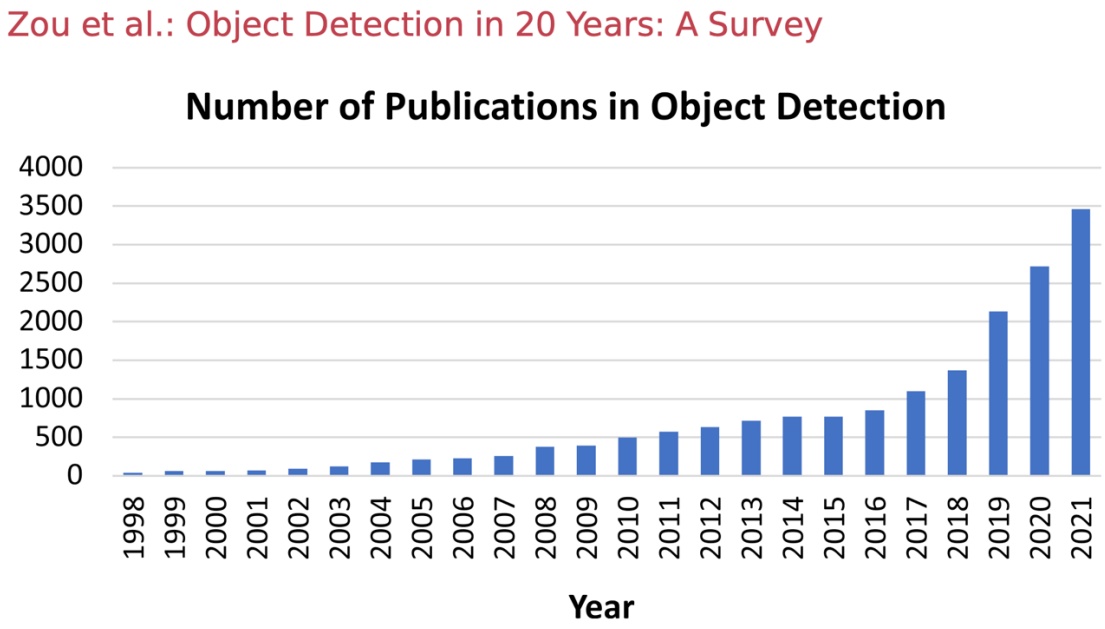


Figure-2: Number of publications in object detection

Among those improvements, traffic object detection has emerged as an important technology within the field of Intelligent Transportation Systems (ITS). Object detection, leveraging advances in machine learning and computer vision, aims to identify, classify, and track various objects, such as vehicles, pedestrians, and other road users, in real-time. By integrating this technology into traffic management systems, cities can optimize traffic flow, reduce congestion, and improve road safety.

In the process of this study, I will be comparatively analyzing several object detection models to determine the most appropriate tool that will serve the purpose of traffic object detection, considering the limitation of my existing processing power and computational expertise. Though Faster R-CNN and SSD are really good at performing their object detection capabilities, among them, YOLO is found to be the most efficient. YOLO manages to realize real-time detection and therefore presents a perfect balance between speed and precision. It is thus especially suitable for small systems and embedded applications. This becomes important, considering that in general, traffic monitoring systems have limited computational resources. Also, regarding efficiency in processing, the architecture of YOLO makes it optimal for this study. (Redmon et al., 2016)

This dissertation explores the implementation of a traffic object detection system that can analyze video footage, detect vehicles, annotate them with bounding boxes, and generate traffic density reports. The system's design leverages deep learning techniques to achieve real-time detection and classification of vehicles, contributing to smarter and safer urban mobility solutions. Moreover, a thorough literature review was conducted to assess the current state of object detection applications for traffic analysis, ensuring that the chosen approach aligns with the best practices in the field.

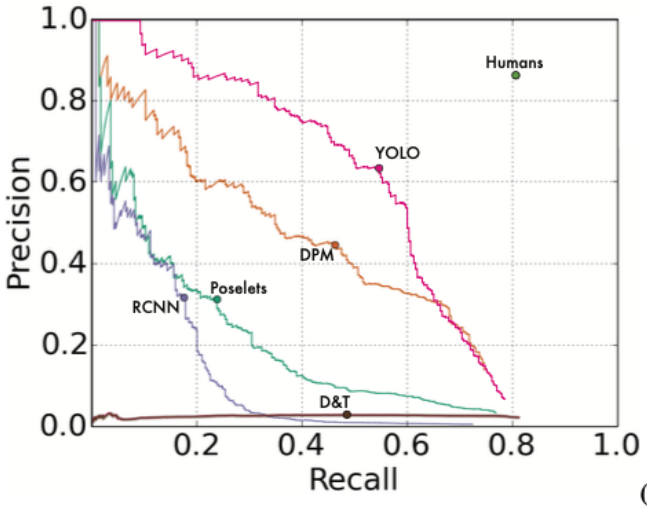
1.1 Motivation

The motivation for this research stems from the growing complexity of urban traffic systems and the need for more efficient management solutions. Traditional traffic control methods often fail to keep pace with the dynamic nature of modern transportation, leading to frequent congestion, accidents, and delays. In cities worldwide, traffic management has become a significant concern for authorities seeking to ensure the safe and efficient movement of people and goods.

Traditional traffic management methods such as static sensors, manual counting, and camera surveillance systems have several limitations, including a lack of real-time adaptability, inefficiency in processing large data streams, and susceptibility to human error. Advances in artificial intelligence (AI), particularly in deep learning, offer a transformative solution by automating the detection and classification of vehicles with unparalleled accuracy and speed. AI models, such as YOLO(You Only Look Once) (Zou et al., 2022), enable the system to not only detect and classify vehicles in real-time but also estimate traffic density dynamically. The use of AI allows for rapid processing of video data and the extraction of actionable insights, empowering authorities to make data-driven decisions for improving traffic flow, reducing congestion, and enhancing urban mobility solutions. Additionally, the scalability of AI-based systems ensures that they can be deployed across various traffic environments, from highways to urban intersections, providing a more adaptable solution than conventional approaches.

Object detection models, particularly those based on deep learning, have demonstrated remarkable success in identifying and tracking objects in various domains, not only on traffic area but also in smart city, heavy indusry, logistics, and retail sector applications . This research aims to harness these technologies to develop a robust traffic object detection system capable of analyzing real-time video feeds and providing actionable insights for traffic management. (Bochkovskiy et al., 2020)

The use of models like YOLOv8, known for their balance of speed and accuracy, provides the opportunity to implement a system that not only detects vehicles but also evaluates traffic density, helping authorities make data-driven decisions to optimize traffic flow and reduce congestion. This research is motivated by the potential of such systems to transform urban transportation networks and improve the quality of life for city residents. (Ultralytics, 2024b)



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure-3: Generalization results on Picasso and People-Art dataset (Redmon et al., 2016)

1.2 Aims and Objectives

The primary aim of this research is to develop a robust traffic object detection system that can analyze given video footage, accurately detect and classify vehicles, and generate traffic density reports for specific time periods. This system will be designed to operate within the constraints of available computational power and be optimized for smaller-scale applications, such as embedded systems or real-time processing on lower-end hardware. To achieve this overarching aim, the following specific objectives have been identified:

1.2.1 **To conduct a comprehensive literature review** of existing object detection models and their applications in traffic management systems. This review will provide insights into the state-of-the-art approaches and help identify the most suitable algorithms for traffic object detection, including YOLO, Fast R-CNN, Faster R-CNN, DETR, and SSD among others. The literature review will also highlight the advantages and limitations of each approach, guiding the selection of the most appropriate model for this research. (Ren et al., 2017)

1.2.2 **To implement a traffic object detection system** that utilizes the YOLOv8 model for real-time detection and classification of vehicles in video footage. YOLOv8 was selected based on its ability to balance detection accuracy and processing speed, making it ideal for real-time applications on systems with limited computational resources. The implementation will involve adapting the YOLO architecture to the specific requirements of traffic object detection, including the customization of model parameters, such as anchor sizes and input image resolution, to optmize performance. (Ultralytics, 2024b)

1.2.3 **To develop a framework for video analysis** that can process uploaded video files, extract relevant frames, and apply the custom trained and pre-trained YOLO models to detect vehicles. As an IDE and coding technology, Goolge COLAB is used for training the mdels, one of the most complex IDEs Intellij is used and PyCharm is the back-end coding insfrastructre of the project. Flask is implemented on the front-end side. This framework will include functionality for annotating detected vehicles with bounding boxes, labeling vehicle types (e.g., car, bus, truck, motorcycle), and providing confidence scores for each prediction. The framework will also be designed to handle log system, that explains and supply the feed-back about back-end mirror processing.

1.2.4 **To design and implement a traffic density reporting mechanism** that summarizes the results of the video analysis. This mechanism will generate reports that include key metrics such as the total number of detected vehicles, the distribution of vehicle types, and traffic density levels across different time intervals. These reports will be presented in both metrics and graphical formats, providing a clear and concise overview of the analyzed traffic data. The reporting mechanism will also include options for exporting the results in the web page, at the different part of the division.

1.2.5 **To evaluate the performance of the developed system** using a set of predefined metrics. This evaluation will involve testing two versions of the YOLOv8n model: the pre-trained model trained on the COCO dataset (Ultralytics, 2024a) and a custom-trained model fine-tuned with a traffic-specific dataset. Key performance metrics will include Mean Average Precision (mAP50), loss function graphs, precision-confidence curves, correlation matrices, and confusion matrices. These metrics will provide a comprehensive assessment of the system's accuracy and efficiency. The evaluation will compare the pre-trained and custom-trained models across various traffic conditions and scenarios to determine which approach is more effective for traffic monitoring.

1.2.6 **To explore potential applications of the developed system in real-world traffic management scenarios.** This objective involves assessing the system's readiness for deployment in specific use cases, such as automated traffic monitoring and traffic flow analysis. Collaboration with relevant stakeholders, such as traffic management authorities, may be pursued to gather feedback and refine the system. However, the primary focus will remain on demonstrating the system's effectiveness in controlled, real-world-like environments rather than full-scale deployment. Any limitations identified during these evaluations will guide further refinements and improvements to ensure the system can meet the practical demands of traffic management.

1.2.7 **To identify areas for future research and development**. An important objective was to document the limitations encountered during the research, particularly challenges with computational resources required for training. We utilized the YOLOv8n version of a 40MB COCO-based pre-trained model (Kaggle, 2024c) and a 180MB traffic-specific subset for custom training (Kaggle, 2024b). Other pre-trained models were integrated into the system for user selection. However, due to the high computational costs on platforms like Google Colab, we were unable to fully train the YOLOv8 medium and large models, which impacted the project's final outcomes. These constraints will be discussed in the Conclusion. Future work may focus on overcoming these challenges by optimizing the system for more powerful hardware or exploring more cost-effective training methods.

2 Literature Review

Overview of Object Detection

Object detection is a foundational and one of the primary tasks in computer vision. It aims to identify and classify objects within images or videos. The field has seen substantial advancements over the past two decades, driven by the increasing availability of large datasets and advances in machine learning, particularly deep learning. Object detection algorithms have evolved from basic methods such as sliding windows and hand-crafted features to more sophisticated approaches using convolutional neural networks (CNNs). Recent approaches have focused on improving both accuracy and speed to enable real-time object detection, which is critical for applications such as autonomous driving and traffic (Redmon et al., 2016).

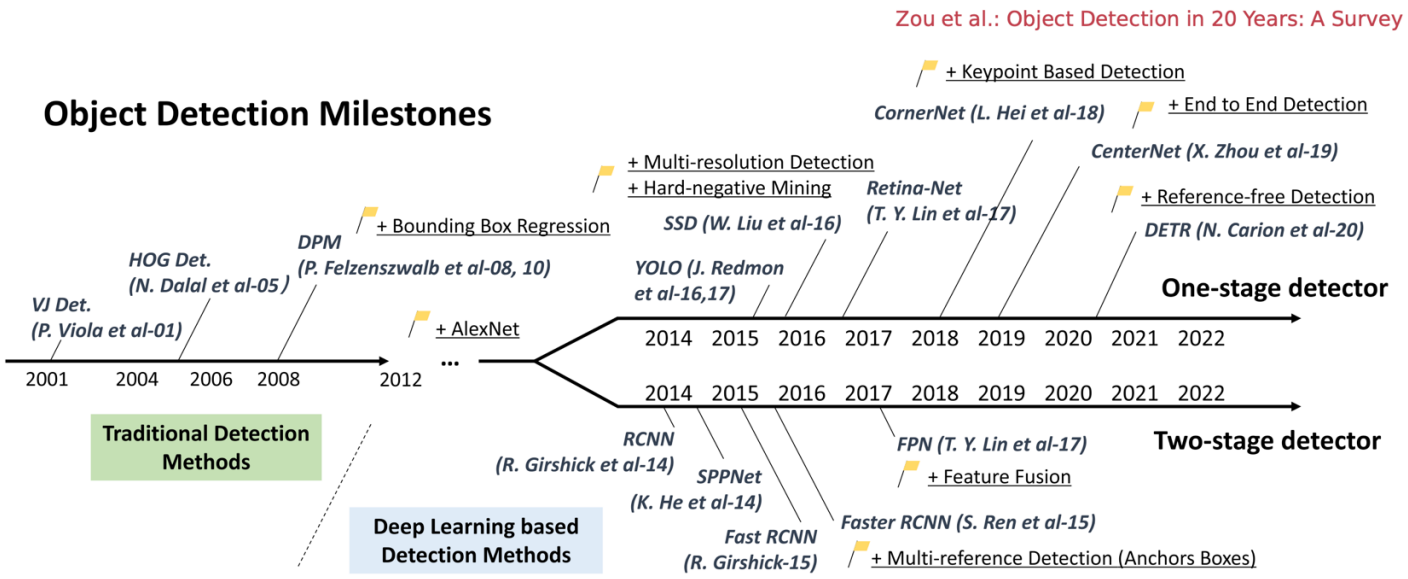


Figure-4: Roadmap of object detection and the milestones detectors by (Zou et al., 2022)

Figure-4, presents an overall roadmap of how object detectors have evolved from traditional detectors such as VJ Det and HOG to state-of-the-art CNN-based modern object detectors: Faster R-CNN, YOLO, and the more recent ones, DETR (Deformable Transformer). This figure summarizes how object detection evolved from handcrafted features, such as HOG and Deformable Part Models, to two-stage CNN-based detectors like R-CNN and its variants, which substantially improved the accuracy of detection but were computationally very expensive. These were followed by one-stage detectors like YOLO and SSD, which further simplified the process of detection by predicting the bounding boxes and their classes directly from the feature maps. This resulted in real-time performance. Very recently, attention-based models such as DETR introduced the use of transformers for object detection, eliminating the need for anchors further and improving the efficiency and performance on complex datasets like MS-COCO. Also, the table below(Table-1) underlines the fast pace of innovation within object detection, where each method tries to address the limitations of the previous one:. With this dissertation, by reusing some YOLO models, it is possible to realize real-time detection of traffic objects that can be applied even in high-speed environments such as highways.

2.1 Region-Based Methods

Region-based methods, such as Faster R-CNN, have been fundamental to object detection. (Ren et al., 2018), Faster R-CNN builds upon earlier models like R-CNN and Fast R-CNN by introducing a Region Proposal Network (RPN) that efficiently generates object proposals. These proposals are then refined and classified using a CNN. This approach significantly reduces the computational overhead compared to previous methods, where object proposals were generated separately, making Faster R-CNN a more viable option for object detection tasks(Ren et al., 2017).

Despite its effectiveness, Faster R-CNN is often limited by its relatively slow inference speed. While it excels in accuracy, making it suitable for high-stakes applications like medical imaging, its performance in real-time scenarios, such as traffic object detection, can be suboptimal. For instance, the Faster R-CNN model can achieve a mean average precision (mAP) of 73.2% but only operates at 7 frames per second (FPS) using the VGG-16 backbone (Ren et al., 2017). This trade-off between speed and accuracy is a key consideration when selecting a model for real-time applications like traffic monitoring.

Comparison Table of Object Detection Models					
Model	**Accuracy**	**Speed (FPS)**	**Suitable for Real-Time**	**Resource Requirement**	**Key Features**
YOLO (You Only Look Once)	High	45	Yes	Moderate	Single neural network predicts bounding boxes and probabilities directly from full images
Faster R-CNN	High	7	No	Very High	Uses a region proposal network for object detection
OpenCV + TensorFlow	Variable	Variable	Yes	Moderate	Combines traditional computer vision with deep learning for flexibility

Table – 1: Object Detection Models Comparison Table

2.2 One-Stage Detectors

One-stage detectors, such as YOLO (You Only Look Once) and SSD (Single Shot Detector), were developed to address the need for faster object detection models. Unlike region-based methods, one-stage detectors eliminate the region proposal step, making them faster and more suitable for real-time applications. YOLO, introduced by Redmon et al. (2016), redefines object detection as a single regression problem, predicting bounding boxes and class probabilities directly from the full image in a single pass through the network. This approach allows YOLO to achieve remarkable speed, processing images at up to 45 FPS, see Table-1, which makes it ideal for traffic object detection systems where quick decision-making is essential.

YOLO has undergone several iterations since its initial release, with YOLOv4 being one of the most popular versions. YOLOv4 improves upon its predecessors by incorporating new techniques such as CSPNet (Cross Stage Partial Network) and Path Aggregation Network (PAN), which enhance both speed and accuracy (Bochkovskiy et al., 2020)These advancements have made YOLOv4 one of the top choices for applications that require high performance in real-time, such as smart traffic systems and autonomous vehicles.

2.3 Lightweight Models and Embedded Systems

With the increasing demand for deploying object detection models on embedded systems, lightweight versions of traditional models have been developed. These models are optimized for environments with limited computational resources, such as roadside cameras and IoT devices. One such model is YOLOv4-tiny, a streamlined version of YOLOv4 that maintains competitive accuracy while significantly reducing the size and complexity of the network (Bochkovskiy et al., 2020).

YOLOv4-tiny is particularly effective in resource-constrained environments where power efficiency and processing speed are critical, making it a suitable choice for traffic object detection in real-world settings.

Similarly, the YOLOv7-tiny model has been proposed for complex road scene detection, emphasizing the need for models that can handle intricate scenarios while running on lower-end hardware(Zhang et al., 2023). These lightweight models strike a balance between speed and accuracy, enabling real-time object detection on devices that cannot support full-scale models like YOLOv4 or Faster R-CNN (Ren et al., 2017; Bochkovskiy et al., 2020).

2.4 Applications in Traffic Object Detection

Traffic object detection is a critical application of object detection models, playing a vital role in intelligent transportation systems (ITS). Effective detection of vehicles, pedestrians, and other road users is essential for improving traffic flow, reducing accidents, and enabling the deployment of autonomous vehicles. YOLO-based models are increasingly being used for these tasks due to their real-time processing capabilities, due its higher number of CNN filter usage,see Figure-5. For example, YOLOv3 and YOLOv4 have been employed in various traffic monitoring systems to detect and classify vehicles, estimate traffic density, and monitor road conditions (Redmon et al., 2016; Bochkovskiy et al., 2020).

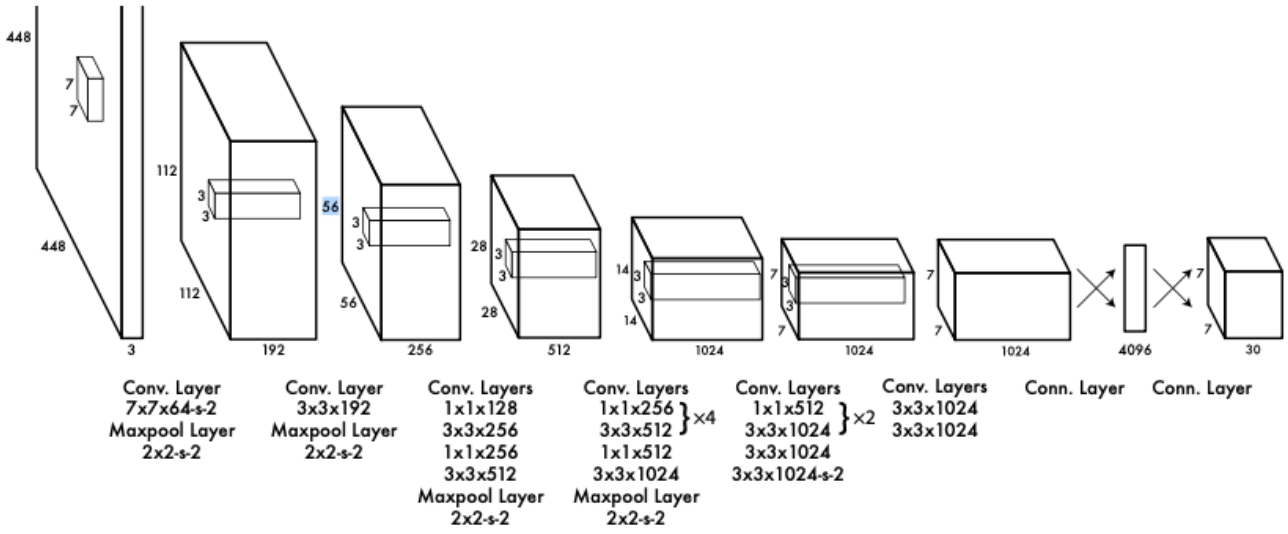


Figure - 5: CNN of YOLO Network Layer: 24 convolutional layers followed by 2 fully connected layers.

Custom-trained models using traffic-specific datasets have shown to outperform general pre-trained models, particularly in domain-specific tasks like traffic monitoring. For instance, a model pre-trained on the COCO dataset can detect a wide range of objects, but retraining the model on a traffic-specific dataset can significantly improve its accuracy in detecting vehicles and other traffic-related objects. Evaluation metrics such as mAP, confusion matrices, and loss function graphs are commonly used to compare the performance of these models, ensuring that they meet the necessary standards for deployment in real-world traffic systems (Ren et al., 2017).

2.5 Strengths and Limitations of Current Models

While models like YOLO and Faster R-CNN have achieved impressive results in object detection, they each have their own strengths and limitations. YOLO's main advantage lies in its speed, making it suitable for real-time applications. However, it tends to struggle with the localization of smaller objects, as the model imposes strong spatial constraints on its predictions (Redmon et al., 2016). In contrast, Faster R-CNN excels in accuracy but is often limited by its slower inference speed, which can be a bottleneck in time-sensitive applications like traffic monitoring (Ren et al., 2017).

Lightweight models like YOLOv4-tiny offer a compromise, providing a balance between speed and accuracy. These models are particularly useful for embedded systems, where computational resources are limited. However, the reduction in network complexity can sometimes lead to a drop in detection accuracy, particularly in complex environments with many overlapping objects (Bochkovskiy et al., 2020).

Choice of an object detection model depends on the specific requirements of the application. For traffic object detection, YOLO-based models are often preferred due to their speed and efficiency, particularly in real-time scenarios. However, for applications where accuracy is more critical than speed, region-based models like Faster R-CNN remain valuable. The ongoing evolution of object detection models, including the development of lightweight versions for embedded systems, continues to expand the potential applications of these technologies in intelligent transportation systems and beyond.

In summary, object detection has seen significant advancements, moving from traditional methods like sliding windows and handcrafted features to deep learning approaches that enable real-time detection. Region-based models such as Faster R-CNN have greatly improved accuracy by introducing a Region Proposal Network (RPN), but their high computational cost limits their applicability in real-time scenarios like traffic monitoring. In contrast, one-stage detectors, particularly YOLO, have simplified the detection process by eliminating the region proposal step, making them ideal for real-time applications with their remarkable speed. Lightweight versions, such as YOLOv4-tiny and YOLOv7-tiny, have further optimized these models for embedded systems, providing a balance between speed and accuracy suitable for resource-constrained environments like traffic object detection. Custom-trained YOLO models, using traffic-specific datasets, have demonstrated superior performance in intelligent transportation systems, enabling efficient detection of vehicles and pedestrians in real-world settings. However, the choice of model depends on the specific requirements, as trade-offs between speed, accuracy, and computational resources remain key considerations.

3 Current Methodologies and Approaches

This section provides a detailed analysis of three notable research papers, followed by a comparative evaluation against the developed system, which focus on traffic object detection and traffic density estimation. These selected papers showcases diverse methodologies for traffic density estimation and object detection, each contributing valuable insights into practical applications for real-world traffic management. The three papers critically examined in this analysis are, (Daisik et al., 2020), (Sahasranand et al., 2021; Mittal et al., 2022).

Through comparing the methods used in these papers along, with their algorithms and datasets to the system being developed here; this study explores the strengths and weaknesses to integration of different approaches to improve the systems effectiveness, in real life situations.

Three papers have been selected based on their significance and different viewpoints, in the area of traffic control and object identification.In the paper,(Daisik et al., 2020), an approach that focuses on data is presented by incorporating details, from connected autonomous vehicles that envision potential traffic systems encompassed by vehicle to vehicle communication. The second research paper, (Sahasranand et al., 2021) authored by Nam Daisik and colleagues, in 2021 discusses a model aimed at improving communication efficiency in distributed systems to boost the scalability of traffic monitoring networks. The third research paper titled “Enhancing Vehicle Detection and Estimating Traffic Volume with a Combination of Deep Learning Models” authored by Usha Mittal and his colleaugues, (Mittal et al., 2022) delves into enhancing the precision of object detection through learning methods. Offers a comparative analysis, with the standalone YOLOv8 model employed in their system development work.These papers collectively offer an exploration of progressions in traffic volume estimation encompassning a spectrum, from empirical methodologies to computational enhancements and advanced machine learning techniques.

3.1 Methodologies and Approaches

Accoring to the methodologies and approaches, aech of the papers under review brings a unique methodological perspective to traffic density estimation and object detection. The developed system in this work take center on the use of YOLOv8 for traffic object detection, particularly it emphase on enhancing labeling accuracy and bounding box stability conditions. These are achieved by implementing processes such as thresholding and smoothing, essential for improving detection performance in complex and dynamic traffic environments.

The third paper, (Mittal et al., 2022), supplies a similar objective but uses an ensemble of deep learning models, including YOLOv4, Faster R-CNN, and SSD (Single Shot MultiBox Detector). By combining these models, the paper addresses the essential trade-offs between speed and accuracy, achieving robust vehicle detection across various traffic conditions. This ensemble method presents a flexible approach, it is capable of adapting to different environmental conditions such as lighting and weather, although it comes at a higher computational cost compared to the single-model approach used in the developed system. While both the developed system and this ensemble method are designed for real-time applications, the focus in the developed work is on computational efficiency through a singular, optimized model (YOLOv8), making it more suitable for environments with limited computational resources, such as Google Colab.

The first paper that is mentioned, (Daisik et al., 2020), takes from image-based detection systems by centering on spatio-temporal data obtained from connected and autonomous vehicles (called CAVs). The system proposed in this paper leverages recurrent neural networks (RNNs) to process temporal data from CAVs, enabling real-time traffic density estimation based on vehicle communication. While the created system is primarily reliant on camera feeds and visual object detection, the CAV-based approach highlights the potential for integrating multiple data sources into traffic density estimation. By processing CAV data with image-based object detection, future iterations of the developed system could significantly enhance traffic insights, particularly in scenarios where camera-based detection might be limited by image blocking issues or other environmental factors.

The second paper, (Sahasranand et al., 2021), paper provides a theoretical exploration of distributed hypothesis testing and correlation testing in high-dimensional data. This is very mathematical and theorical stuy and the reason of placing this to this research, is to put that research on the spot in the improvement areas of traffic detection and density studies. Study is more on minimizing communication complexity between distributed systems, which could be particularly relevant to traffic management systems operating across multiple locations. While the developed system does not directly address the communication aspect, this paper's insights could be applied to optimize real-time data transfer between traffic detection systems operating in large-scale urban environments. In real-world traffic management scenarios, where actually bandwidth limitations and data latency may pose significant challenges, reducing communication overhead could enable faster and more efficient traffic data processing. As such, the developed system could benefit from integrating the theoretical frameworks proposed in this paper to enhance its scalability and performance across distributed networks.

3.2 Algorithms and Computational Approaches

From an algorithmic perspective, current system and (Mittal et al., 2022) leverage YOLO for object detection. In the developed work, YOLOv4 or later models are selected for its balance of speed and accuracy, optimizing it specifically for real-time applications. The third paper, on the other hand, uses YOLOv4 in combination with Faster R-CNN and SSD, forming an ensemble that maximizes detection performance across various conditions. The method offers higher adaptability, as each model within the ensemble excels in different scenarios, whether it be precision or speed. However, this comes at the cost of greater computational resources, as multiple models must be run in parallel to achieve the desired performance.

The developed system that is presented, on the other hand, prioritizes computational efficiency by focusing on a single, lightweight model (YOLOv8). This choice is justified in environments where computational resources are limited, such as the Google Colab platform used during the development phase. The decision to focus on YOLOv8 enables the system to operate in real-time with minimal latency, making it ideal for large-scale deployments in smart cities and other urban settings where high-speed traffic monitoring is essential.

In contrast, the algorithmic approach of (Daisik et al., 2020) is fundamentally different, as it uses RNNs to process temporal traffic data. This approach is suited for capturing traffic flow patterns over time, leveraging the continuous data streams generated by CAVs. While the RNN-based model improved at temporal predictions, its application in visual object detection would be limited. Yet, integrating the temporal insights from this method with the real-time visual detection capabilities of YOLOv8 could offer a more holistic view of traffic conditions. Then, combines both spatial and temporal data to deliver more accurate and comprehensive traffic management solutions.

3.3 Datasets and Real-World Applications

The datasets employed in each of these studies reflect the varied approaches to traffic density estimation and object detection. The developed system uses a custom traffic dataset comprising 21 classes, including various vehicle types, for training YOLOv8. The dataset includes real-time video feeds, enabling the system to be tested and optimized for real-world deployment, with a focus on urban and highway settings. This real-world applicability is essential for traffic management systems, as it ensures the model is exposed to diverse environmental conditions and traffic dynamics.

Similarly, (Mittal et al., 2022) utilizes real-world traffic datasets, such as KITTI, to train and evaluate its ensemble of models. KITTI is a less popular public database comparing COCO which is used in proposed system. These datasets, which include labeled images of vehicles captured under varying weather and lighting conditions, are crucial for assessing the robustness of the ensemble model. Both the developed system and the ensemble-based approach are designed to function in real-world environments, making them well-suited for applications such as autonomous driving, congestion monitoring, and urban traffic control. However, the ensemble method may offer higher accuracy in detecting objects under challenging conditions, while the developed system maintains a focus on achieving real-time performance in resource-constrained environments.(See Table-2)

Model	Dataset	mAP	CY	TW	LV	HV	TR	BU
SSD	FLIR RGB	0.58	0.65	0.54	0.64	0.55	0.58	0.5
	FLIR Thermal	0.65	0.7	0.64	0.61	0.58	0.81	0.55
	KITTI	0.60	0.55	0.51	0.51	0.59	0.46	0.93
	MB7500	0.52	0.52	0.47	0.59	0.41	0.57	0.56
Faster R- CNN	FLIR RGB	0.87	0.94	0.87	0.8	0.94	1	0.67
	FLIR Thermal	0.88	0.96	0.91	0.86	0.87	0.86	0.8
	KITTI	0.81	0.75	0.75	0.84	0.89	0.77	0.85
	MB7500	0.68	0.68	0.6	0.59	0.94	0.6	0.67
Proposed Ensemble	FLIR RGB	0.92	0.94	0.93	0.91	0.91	1	0.83
	FLIR Thermal	0.94	0.95	0.96	0.89	0.97	1	0.9
	KITTI	0.92	0.85	0.87	0.96	0.92	0.97	0.98
	MB7500	0.86	0.89	0.8	0.96	0.94	0.78	0.78

Table-2: Comparative analysis of SSD, Faster R-CNN and KITTI, and MB7500 Proposed ensemble based upon mAP on FLIR.

In contrast, (Daisik et al., 2020) uses data from CAVs rather than visual datasets. This futuristic approach relies on real-time communication between vehicles, enabling the system to estimate traffic density based on vehicle-to-vehicle (V2V) data. While this method gives a more direct measure of traffic density than image-based detection, it limites by the current penetration rate of CAVs in traffic networks. In regions where CAVs are not yet prevalent, image-based object detection systems like the developed system will remain essential for traffic monitoring. Nonetheless, as the adoption of CAVs increases, integrating this data with image-based detection could create more sophisticated traffic management solutions that leverage both visual and CAV data for more accurate and responsive traffic density estimation.

3.4 Evaluation Metrics and Results

Both the developed system and (Mittal et al., 2022) use common evaluation metrics called ‘precision’, ‘recall’, and ‘mean Average Precision’ (mAP) to assess object detection performance. These metrics are critical for ensuring the accuracy and reliability of real-time traffic detection systems, where false positives or missed detections could have serious consequences in traffic management decisions. The developed system’s focus on improving the stability of bounding boxes and optimizing confidence thresholds is a notable contribution, as it directly addresses one of the primary challenges in real-time object detection—maintaining accuracy without compromising speed.

The third examined paper, while also focusing on precision and recall, further emphasizes computational efficiency metrics, given the increased resource demands of its ensemble method. This balance between accuracy and computational cost is particularly relevant to real-world applications, where systems must operate in resource-constrained environments. By comparison, the developed system’s use of YOLOv8 ensures that the system remains lightweight and efficient, making it ideal for deployments in smart city applications where computational resources may be limited.

(Daisik et al., 2020) employs different metrics, primarily focusing on ‘Mean Squared Error’ (MSE) to assess the accuracy of traffic density predictions. While the evaluation metrics differ from those used in object detection, the underlying goal remains the same: ensuring the system provides accurate, real-time estimates in dynamic traffic environments. The CAV-based approach offers a novel perspective on traffic density estimation, which could complement the object detection capabilities of the developed system by providing additional, non-visual data points to enhance traffic flow predictions.

3.5 Strengths, Limitations, and Future Directions

Both the developed system and (Mittal et al., 2022) demonstrates strengths in their ability to detect vehicles in real-time under diverse traffic conditions. The developed system’s emphasis on improving bounding box stability and accuracy within a single model is particularly notable, as it addresses a key challenge in real-time object detection. Current approach, while offering bigger adaptability, comes with increased and greater computational demands, which may limit its applicability in resource-constrained environments. Future iterations of the developed system could explore the integration of ensemble techniques to enhance detection accuracy without sacrificing computational efficiency.

In terms of limitations, the developed system may face challenges in environments with highly complex traffic conditions, such as severe weather or heavy occlusion. Similarly, the method’s computational requirements could hinder its scalability in large-scale traffic networks.

To sum up, the analysis of these three papers reveals different approaches to traffic density estimation and object detection. While current delivered system emphasizes real-time performance -by a particular time delay- and computational efficiency, the ensemble-based method in (Mittal et al., 2022) offers superior adaptability at the cost of increased computational demand, which is ahead of current delivered system. The futuristic CAV-based approach described in (Daisik et al., 2020) provides a novel vision for traffic monitoring, which could eventually supplement visual detection systems as CAV adoption increases. That can be thought in different domains using different type of data but serves similar goals. Together, these studies highlight the potential for integrating multiple approaches to create more robust and scalable traffic management solutions.

4 Implementation of Object Detection System

4.1 What is YOLO

Object detection has evolved from early handcrafted based methods, including HOG-Histogram of Orientied Gradients- and the Viola-Jones detector, to modern deep learning approaches. In the last twenty years, the technology of object detection has rapidly developed with continuous breakthroughs in CNNs, culminating in the state-of-the-art models: Faster R-CNN, SSD, and YOLO. The shift from feature-based methods to deep learning approaches has enabled better accuracy, adaptability, and speed, making CNNs the cornerstone of object detection systems today (Zou et al., 2022). These deep learning-based object detection systems automatically extract hierarchical features from images, significantly improving performance across various domains such as traffic monitoring, autonomous driving, and surveillance systems.

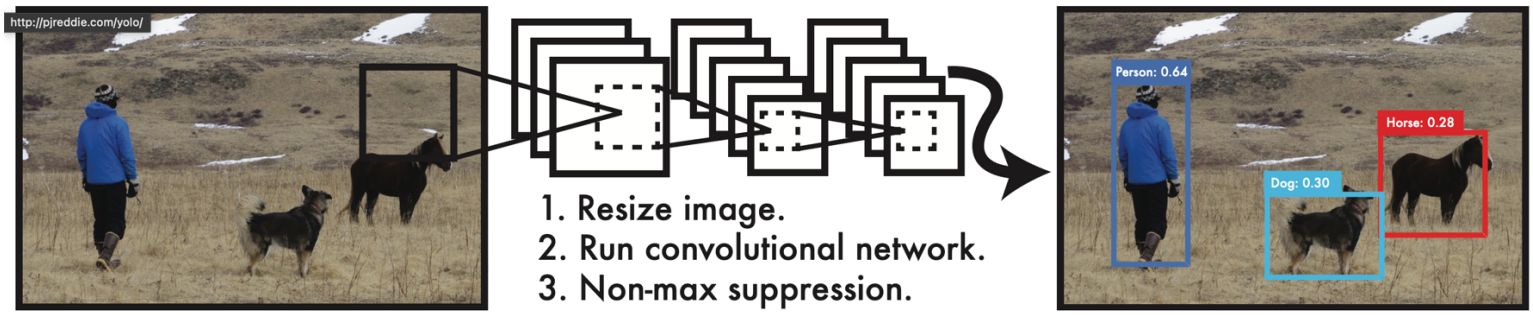


Figure-6: YOLO Resizing Model - YOLO detection system resizes the input image to 448 × 448, applies a convolutional network, and filters detections based on confidence thresholds.

YOLO was first proposed by -Redmon et al. in 2016- and is one of the most impactful object detection models that employs CNNs for achieving real-time detection. YOLO achieves this by using a single neural network to perform both object localization and classification. Unlike earlier models, which treated object detection as a pipeline of separate stages, YOLO redefines it as a regression problem, enabling fast and efficient detection without sacrificing accuracy (Redmon et al., 2016). In CNN-based architectures, especially in YOLO, convolutional layers play a critical role by extracting key features from the input image, such as edges, textures, and object shapes. The deeper the layers of the CNN, the more abstract the features extracted would be, which thus enables distinguishing various objects in the image.

In YOLO, the input image goes through a sequence of convolutional layers in order to generate feature maps. These feature maps provide the foundation upon which YOLO builds its object detection pipeline. It divides the input image into an S × S grid; each grid cell is responsible for detecting objects whose center falls within it. For each of these grid cells, YOLO predicts bounding boxes, confidence scores, and class probabilities.(See Figure-7)

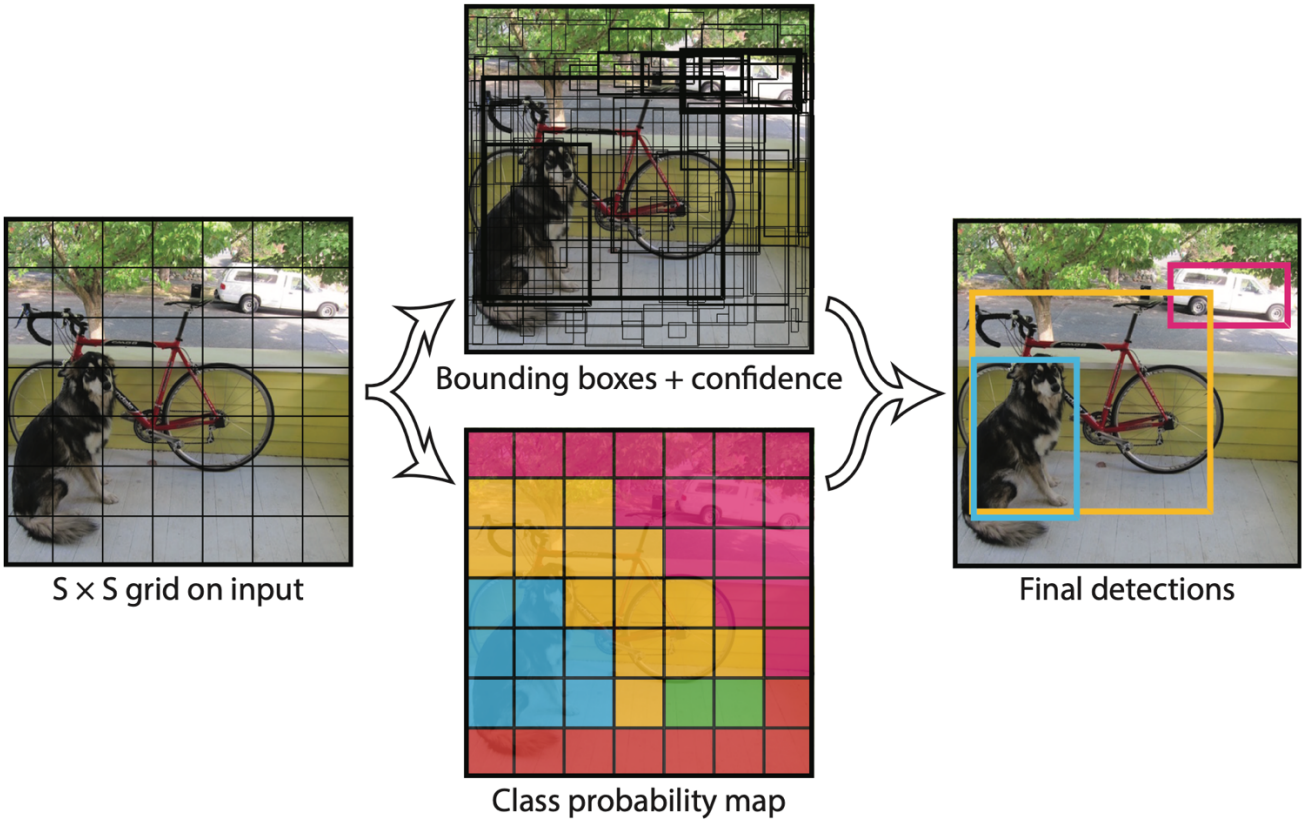


Figure-7: S×S grid, predicting B bounding boxes, confidence scores, and C class probabilities for each cell, resulting in an S × S × (B * 5 + C) tensor.

Bounding boxes in YOLO are settled four parameters: the x and y coordinates of the box center, and the width and height of the box, all normalized relative to the grid size (Bochkovskiy et al., 2020). To guess objects of varying sizes and aspect ratios, YOLO uses anchors—predefined bounding box shapes—enhance localization accuracy. YOLOv4 incorporates several optimizations that improve both speed and accuracy. So, the backbone network CSPDarknet53 reduces computational complexity without sacrificing performance. Additionally, Spatial Pyramid Pooling (SPP) layers are integrated to enhance multiscale object detection by pooling information at different levels of the feature maps, significantly improving the detection of small objects over large images—a challenge for earlier models (Bochkovskiy et al., 2020). Furthermore, the use of Path Aggregation Network (PANet) improves feature pyramid aggregation by enabling better information flow from lower to higher layers, enhancing small object detection capabilities (Ren et al., 2017).

The primary advantage of YOLO's convolutional neural network (CNN) architecture is its grid-based approach to bounding box prediction. By dividing the input image into a grid, each cell predicts a set of bounding boxes accompanied by confidence scores that estimate both the likelihood of an object's presence and the accuracy of the bounding box relative to the object. YOLO optimizes its loss function using the Intersection over Union (IoU) metric to quantify the overlap between predicted bounding boxes and the ground truth. Advanced IoU variants, such as Complete IoU (CIoU) and Distance IoU (DIoU), were incorporated into YOLOv4 to enhance model convergence and improve the precision of bounding box predictions (Bochkovskiy et al., 2020). Furthermore, Non-Maximum Suppression (NMS) is utilized to eliminate redundant bounding boxes, retaining only the highest-quality prediction for each object.

Understanding the historical development, early approaches like Faster R-CNN and Single Shot MultiBox Detector (SSD) were innovative but computationally intensive due to their reliance on region proposal stages (Ren et al., 2017). Contrary, YOLO simplifies the detection pipeline by eliminating the need for explicit region proposals, reformulating object detection as a regression task that predicts object locations and classes directly from the image. This architectural choice significantly accelerates detection speed, making YOLO highly suitable for real-time applications such as traffic monitoring, where immediate vehicle detection is critical for congestion management and accident prevention (Zhang et al., 2023).

This project employs YOLOv8, the latest iteration developed by Ultralytics. YOLOv8 offers enhanced capabilities, including instance segmentation, pose/keypoint estimation, and classification. Building upon previous advancements, it delivers cutting-edge performance in terms of accuracy and speed, making it an ideal choice for diverse object detection tasks across various applications (Ultralytics, 2024b).

Implementing sign detection using YOLOv8 holds significant potential in practical applications. It can greatly enhance traffic management systems by efficiently detecting and recognizing traffic signs, thereby improving road safety through accurate interpretation and response to sign information. Additionally, it can assist in urban planning and infrastructure development by analyzing the presence and condition of signs in different areas (Kaggle, 2024a). This project aims to harness YOLOv8's capabilities to develop a reliable sign detection solution with the potential to improve various domains that rely on precise and efficient sign identification (Ultralytics, 2024a).

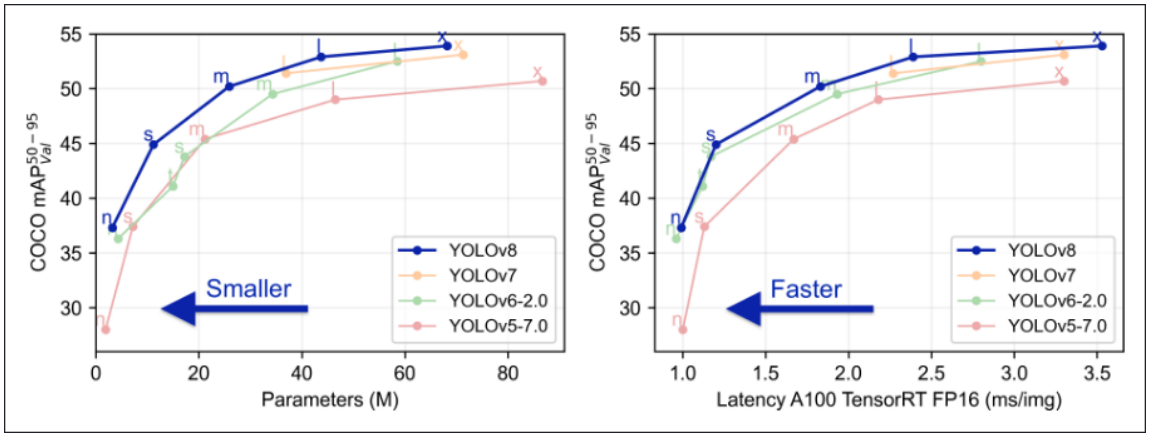


Figure-8: YOLO Versions Comparison Tables

4.2 YOLO Version Selection

4.2.1 YOLOv8 Pre-trained Models and Selection for Traffic Object Detection

The YOLOv8 suite offers a range of pre-trained object detection models, specifically trained on the Common Objects in Context (COCO) dataset. The COCO dataset is one of the most comprehensive datasets in computer vision, encompassing 80 object categories that span a wide variety of everyday objects. It is designed for tasks such as object detection, segmentation, and image captioning, making it an ideal dataset for training general-purpose object detection models. (Ultralytics, 2024b)

YOLOv8 introduces five model variants: nano (YOLOv8n), small (YOLOv8s), medium (YOLOv8m), large (YOLOv8l), and extra-large (YOLOv8x). These models vary in complexity, performance, and computational requirements. As seen in the provided data (Table-1), a clear trend emerges where larger models exhibit higher mean Average Precision (mAP) values, which correlate with increased detection accuracy. However, this improved accuracy comes at the cost of reduced inference speed. For instance, the YOLOv8x model achieves the highest mAP of 53.9%, but its speed on CPU (ONNX) is significantly slower, with an inference time of 479.1 milliseconds (ms). Conversely, the YOLOv8n model, which is the smallest and fastest variant, achieves an inference speed of 80.4 ms on CPU, but with a relatively lower mAP of 37.3% .

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table-3: YOLOv8 Models Performance Comparison Table

All YOLOv8 models are trained on images with a resolution of 640x640 pixels, a standard input size that ensures a balanced trade-off between accuracy and computational efficiency across various applications. This resolution is particularly useful in scenarios where both performance and real-time processing are crucial.

For this particular project, traffic detection and density estimation application, YOLOv8 nano pre-trained model (yolov8n.pt), YOLOv8 medium pre-trained model (yolov8m.pt), and YOLOv8l (yolov8l.pt), and customed trained nano models, are selected to handle vehicle detection.

4.2.2 Evaluation Metrics: Intersection Over Union (IoU) and Mean Average Precision (mAP)

To systematically evaluate the accuracy of the object detection and traffic density estimation components, several key performance metrics are used. Two key evaluation metrics are used to assess the performance of object detection models: Intersection Over Union (IoU) and Mean Average Precision (mAP). IoU is a fundamental metric that evaluates the overlap between the predicted bounding box and the ground truth bounding box. The IoU score ranges from 0 (no overlap) to 1 (perfect overlap). In practice, an IoU threshold (e.g., 0.5 or 0.75) is commonly applied to determine whether a detection should be considered a true positive or a false positive. This threshold plays a crucial role in calculating precision and recall, which are subsequently used to compute the Average Precision (AP) for each object class. (Ultralytics, 2024b)

Mean Average Precision (mAP) is a widely adopted metric for evaluating the overall precision of object detection models. mAP is calculated as the average of the AP values across all object classes. It reflects the model's ability to balance precision and recall, providing a comprehensive measure of the model's performance in detecting all relevant objects with high accuracy. For the evaluation of YOLOv8 models, mAP values are calculated across different IoU thresholds, typically ranging from 0.5 to 0.95, providing a nuanced view of the model's detection capabilities.(Ultralytics, 2024b)

4.2.3 Model Selection for Traffic Object Detection and Traffic Density Estimation

For the specific application of object detection and traffic density estimation, the selection of the object detection model must judiciously balance the trade-off between accuracy and inference speed. Given the stringent computational demands of real-time systems, the YOLOv8n model was chosen for vehicle detection due to its minimal architectural complexity, optimized for high-speed inference. While the YOLOv8n variant achieves the fastest frame processing time among the YOLOv8 models, making it highly fit for real-time traffic monitoring, its nature comes at the cost of reduced accuracy, with a mean Average Precision (mAP) of 37.3%. This is significantly lower compared to its larger competitors, such as the YOLOv8 Large, which, although more accurate, demonstrates slower performance with an inference time of 80.4 milliseconds per frame on GPU hardware. The choice of YOLOv8n in this project partially satisfies the real-time processing demands but imposes limitations on detection precision, particularly when applied to the COCO-trained dataset. Despite these constraints, the YOLOv8n model facilitated the training of two custom datasets, one with 21 traffic-specific classes and another focusing on a single class, enabling a comparative evaluation against the medium and large YOLOv8 models. This experimental setup underscores the trade-offs inherent in selecting a model that balances real-time performance and detection fidelity in traffic analysis applications.(Ultralytics, 2024a)

This selection balances the trade-off between accuracy and speed, and cost prioritizing the system's ability to function in real-time environments, which is critical for dynamic applications such as traffic density estimation. If there was no cost limitation, project could be tested bu 4-5 days, total of more than 120 hours of training for the YOLOv8 medium or large models. Then, probably the occuracy and then the quality of the output program would have significant differentiation from the current one. Thus, future iterations of this system may explore the integration of larger YOLOv8 models should computational resources permit, allowing for enhanced detection accuracy without compromising real-time performance.(Ultralytics, 2024a)

Example of Pre-trained YOLOv8n traffic object detection output frame:

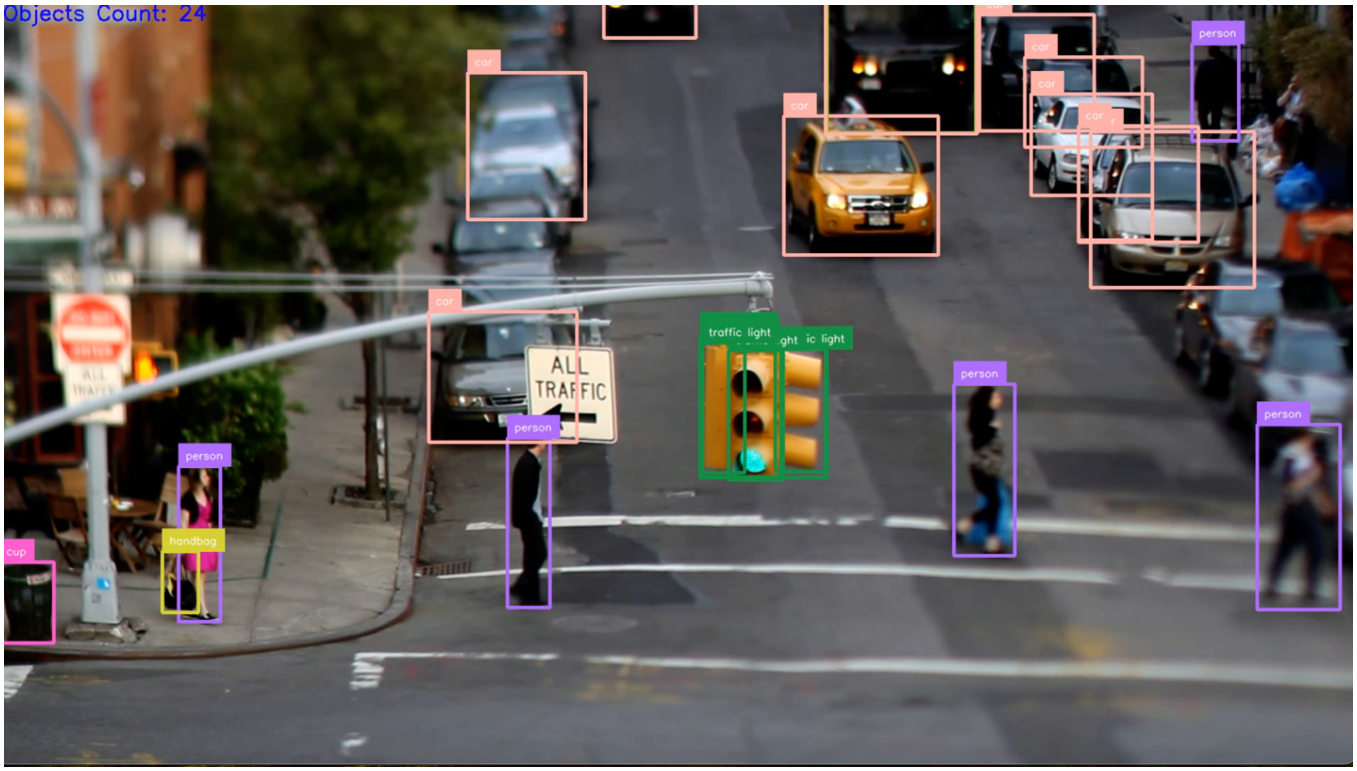


Figure-9: YOLOv8 Nano Pre-trained Model Output Video Frame

4.2.4 YOLOv8 Nano Custom Trained on a 21-Class Traffic Dataset

To implement the traffic object detection and density estimation system, one of the models is YOLOv8 Nano (YOLOv8n) model was selected for custom training, considering its balance of speed and computational efficiency, which are crucial for real-time traffic density estimation. While writer computational power, dedicated time and budget is limited, the whole 19Gb COCO Dataset training is not applicable in those circumstances. Tnen, the custom training process was conducted using a traffic-specific dataset obtained from Kaggle, comprising 185 MB of data across 21 distinct classes such as buses, trucks, cars, and motorcycles. The training was performed on Google Colab Pro, a platform providing limited computational resources with T100 GPU processing unit for limited time, which influenced the model selection and the training duration.. Given the constraints, the YOLOv8n model was chosen for custom training, as training the larger YOLOv8 medium or large variants would have required a significantly longer time (approximately 4-5 days) and higher costs. The training process for the YOLOv8n model took around 14 hours, reflecting a compromise between achieving sufficient detection accuracy and adhering to the project's budgetary and time limitations.

Before the data is fed into the YOLO model for detection, preprocessing steps are applied to ensure optimal performance. The raw traffic videos are first resized to a standard resolution of 640x640 pixels, which is the input resolution required by the YOLOv8 models. Additionally, any noise in the video feed, such as glare or reflection from streetlights, is minimized using contrast enhancement techniques. The dataset is then annotated manually to ensure accurate bounding boxes and labels for the objects, using tools such as Label-Img for custom-trained models. These preprocessing steps ensure that the model receives clean, well-annotated data, thus improving its detection performance, particularly in challenging lighting and weather conditions

The shown table in below (Table-4), is an outcome of the detailed performance summary for the custom-trained YOLOv8n model across 21 distinct traffic-related classes. The table presents several critical metrics, such as precision (P), recall (R), mAP50, and mAP50-95, offering insights into the model's detection performance on the validation set. The overall precision across all classes is 0.584, indicating the model's ability to classify true positives in medium/low level among the detected objects. However, the recall is relatively lower at 0.418, reflecting that the model may miss some of the actual objects in the scene, particularly for more challenging or underrepresented classes such as "minibus" (R = 0.50) and "minivan" (R = 0.34). The model processes each image in approximately 39.3 ms, making it suitable for real-time traffic detection applications, where rapid inference is critical. However, the code of the application should reflect this parrallel processing of reading and processing simultaniously. That's another challenge for the further applications. The model shows strong performance in detecting frequently occurring classes like cars (P = 0.78), buses (P = 0.80), and rickshaws (P = 0.68), with high precision and competitive mAP50 scores of 0.771, 0.655, and 0.707, respectively. This demonstrates its ability to accurately detect and classify common traffic objects.

```
Ultralytics YOLOv8.2.76 🚀 Python-3.10.12 torch-2.3.1+cu121 CPU (Intel Xeon 2.20GHz)
Model summary (fused): 168 layers, 3,009,743 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/traffic_data/valid/labels.cache... 300 images, 0 backgrounds, 0 co
val: WARNING ⚠ /content/traffic_data/valid/images/Pias--360-_PNG.rf.8405b0e44009a9300e0a1

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                   all        300       2568      0.584      0.418       0.49      0.287
               bicycle         30         32      0.519      0.505      0.526      0.236
                   bus        220        425        0.8      0.513      0.655      0.432
                   car        232        842       0.78      0.709      0.771      0.518
               minibus          2          2      0.306        0.5      0.501      0.399
               minivan         87        110      0.407      0.336      0.335      0.243
             motorbike        166        335      0.629      0.534      0.514      0.183
                pickup        105        142      0.509      0.282      0.337      0.184
             policecar          1          1          1          0          0          0
              rickshaw         62        192      0.682      0.681      0.707      0.447
               scooter          1          1          1          0      0.995      0.298
                   suv         49         60      0.219      0.283      0.149        0.1
                  taxi         18         19      0.939      0.812      0.816      0.549
  three wheelers -CNG-        148        252      0.759      0.679      0.735      0.495
                 truck         53         84      0.519      0.571      0.523      0.343
                   van         52         62      0.273       0.29      0.212       0.14
            wheelbarrow          9          9          0          0      0.063     0.0249
Speed: 0.3ms preprocess, 39.3ms inference, 0.0ms loss, 2.3ms postprocess per image
Results saved to runs/detect/val
                     Metric Value
metrics/precision(B)       0.58377
metrics/recall(B)          0.41849
metrics/mAP50(B)           0.48997
metrics/mAP50-95(B)        0.28702
fitness                    0.30732
```

Table-4: 21-Class Custom Trained Model Performance Table

As a clear weakness of the training output, it can be said that for less common or more complex vehicle types, such as police cars (mAP50 = 0) and scooters (mAP50 = 0.995, though with just one instance), the model struggles with accurate detection. This is likely due to an imbalance in the training dataset or difficulty in distinguishing these vehicles within the traffic flow. Also, The model's mAP50-95 score, which measures precision across a range of IoU thresholds, is relatively low at 0.287. This suggests that the model's bounding box predictions are not consistently tight across classes, affecting its accuracy in situations that demand higher IoU thresholds. Apart from the class imbalances and limited generalization, training model also has complex class variability. Certain vehicle types, such as SUVs (P = 0.21, R = 0.28) and vans (P = 0.27, R = 0.29), have lower precision and recall. This might be due to the visual similarity between these classes and other vehicle types, making it harder for the model to distinguish between them.

Following the training, the model's performance was evaluated using key metrics such as mean Average Precision (mAP), Intersection Over Union (IoU), and the F1-Confidence Curve which is shown at Figure-10. The confusion matrix generated from the results provided a detailed insight into the model's predictive accuracy across the 21 traffic classes. For instance, the model demonstrated strong recall for classes like 'car' (0.71) and 'rickshaw' (0.70), but struggled with others such as 'bicycle' (0.51) and 'minivan' (0.55), indicating areas for potential improvement. The biggest challenge of the model is the overall recall of the model is 0.418, indicating that the model is missing a significant number of true positives, especially in certain classes. For instance, pickup trucks (R = 0.28) and minivans (R = 0.34) show relatively poor recall, meaning the model often fails to detect these objects.

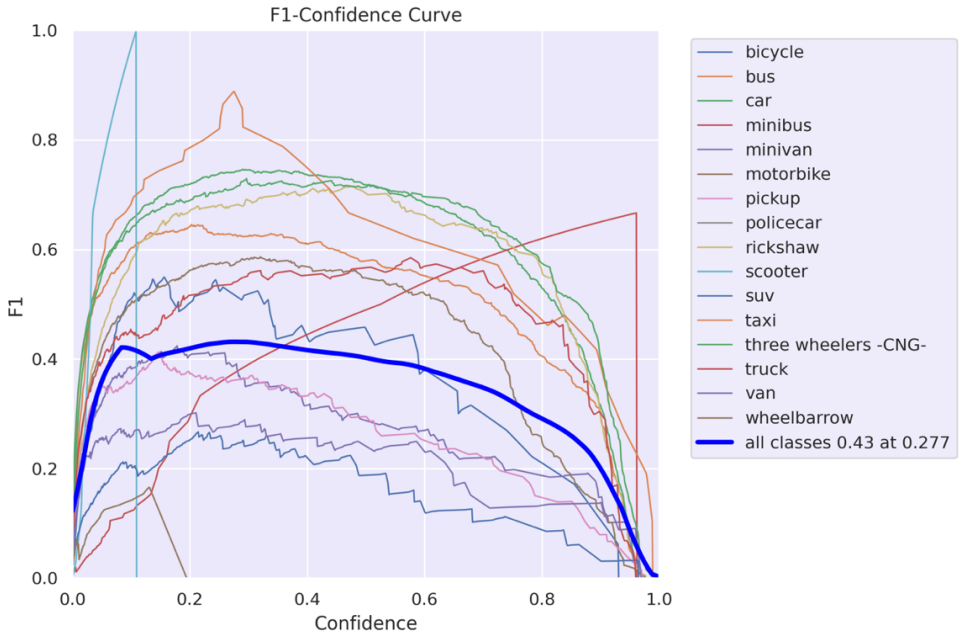


Figure-10: 21-Class Custom Trained Model F-1Confidence Curve

The F1-Confidence and Precision-Confidence Curves highlighted the trade-offs between precision and recall at various confidence levels, with the model's overall F1 score peaking at 0.43. This suggests that while the YOLOv8n model can provide balanced performance, there is room for further optimization, particularly in minimizing false positives(Figure-10, and 11).

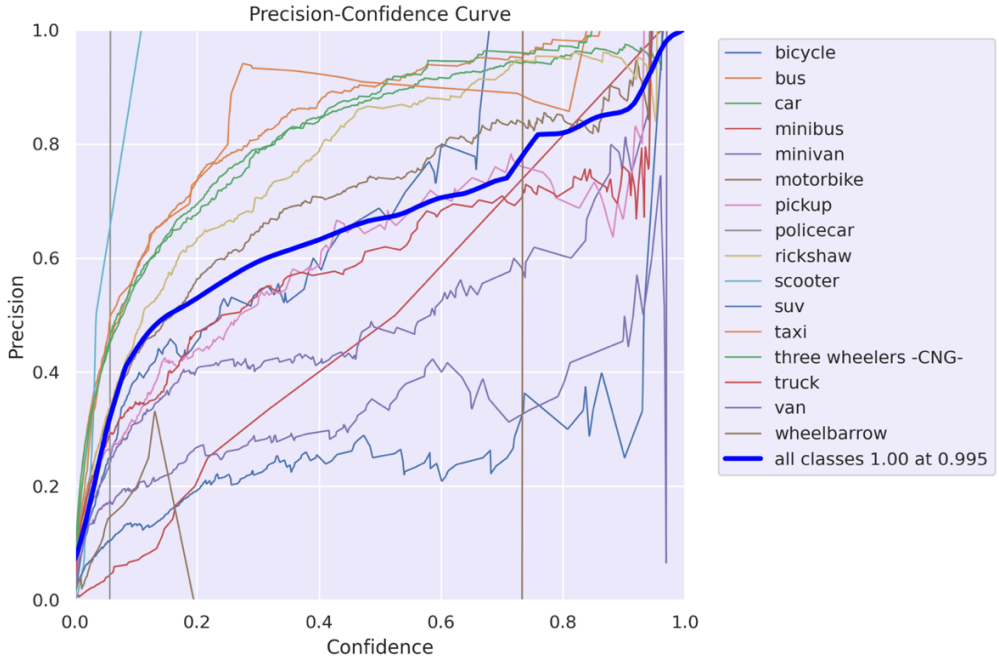


Figure-11: 21-Class Custom Trained Model Precision-Confidence Curve

The training and validation loss curves further underscored the model's learning trajectory, showing a consistent reduction in loss over the 100 training epochs. The final mAP50 achieved was 0.489, with an mAP50-95 of 0.287, indicating that the model was able to generalize effectively to the custom traffic dataset. However, these results also point to the potential for improved performance with extended training or more powerful models.

The normalized confusion matrix at Figure-12, visualizes the performance of the YOLOv8n model on the traffic-specific dataset by comparing the true labels with the predicted labels across all 21 classes. Notably, the diagonal elements indicate the accuracy of the model for each class. High accuracy is observed for classes such as 'car' (0.71), 'rickshaw' (0.70), and 'minibus' (1.00), suggesting that the model performs well in identifying these vehicles. However, other classes like 'bicycle' (0.50) and 'motorbike' (0.59) show moderate performance, with several off-diagonal elements indicating misclassifications. For example, the 'bus' class often overlaps with 'car' and 'minivan', indicating that these vehicles are sometimes confused with each other, potentially due to similar visual features in the dataset. Overall, the matrix highlights areas where the model excels and where further refinement or additional data might be needed to improve accuracy.

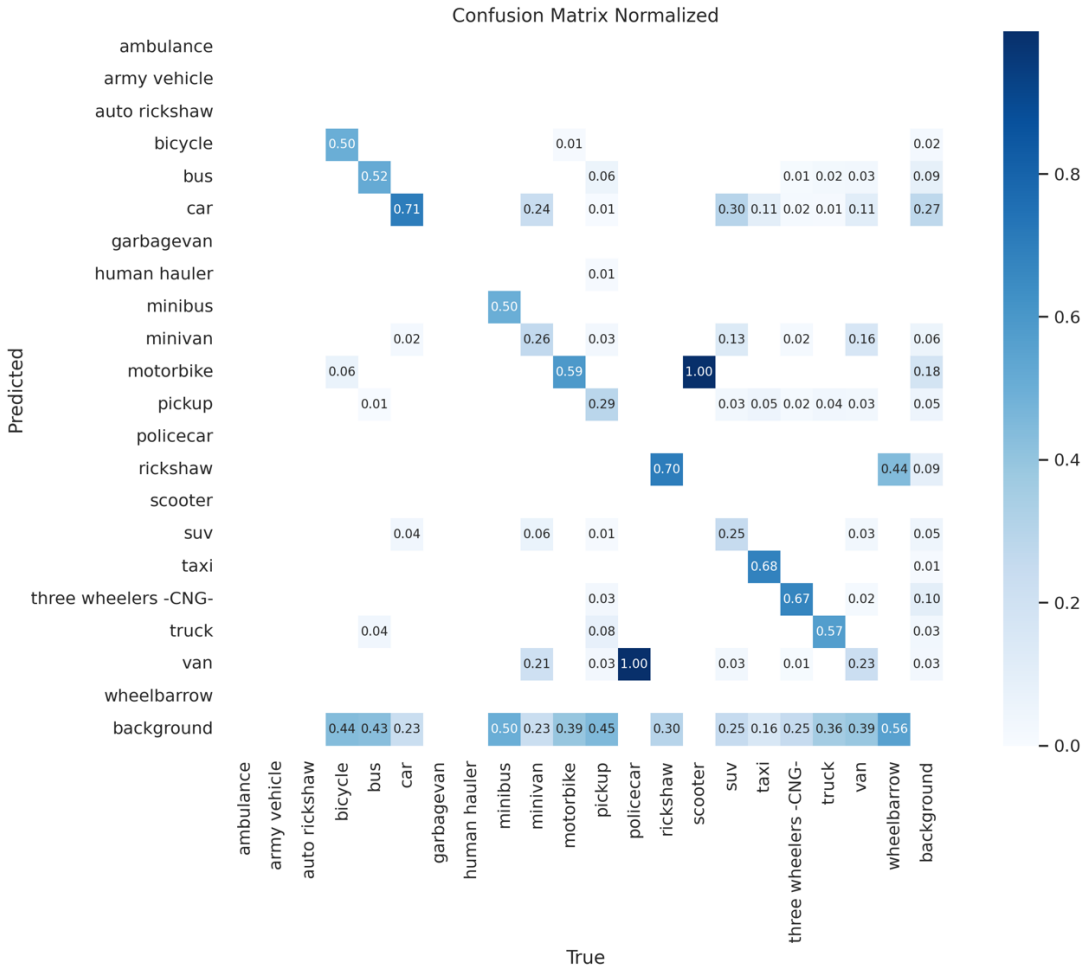


Figure-12: 21-Class Custom Trained Confusion Matrix

The class distribution histogram at figure in below,(Figure-13) from the custom-trained YOLOv8n model on the 21-class traffic dataset reveal several key insights into its performance and the characteristics of the dataset. It shows a significant imbalance, with classes like 'car' and 'rickshaw' having the highest representation, which likely leads to better detection accuracy for these classes, while underrepresented classes like 'policecar' and 'scooter' may suffer from poorer performance. The bounding box visualization indicates a central clustering of objects, suggesting that most objects in the dataset are located near the center of the images, which could bias the model's generalization to new data where objects are not centrally placed. Additionally, the scatter plots of object locations and bounding box dimensions highlight that most objects are of moderate size and centered in the frame, with fewer instances of very large or small objects. This distribution suggests the model may perform well on medium-sized objects but could struggle with extremes in object size. The overlap of bounding boxes also indicates potential challenges in distinguishing closely situated objects, possibly leading to misclassifications. These observations underscore the importance of addressing class imbalance, central bias, and optimizing the model's handling of varying object sizes to enhance overall detection accuracy and robustness in real-world traffic monitoring applications.

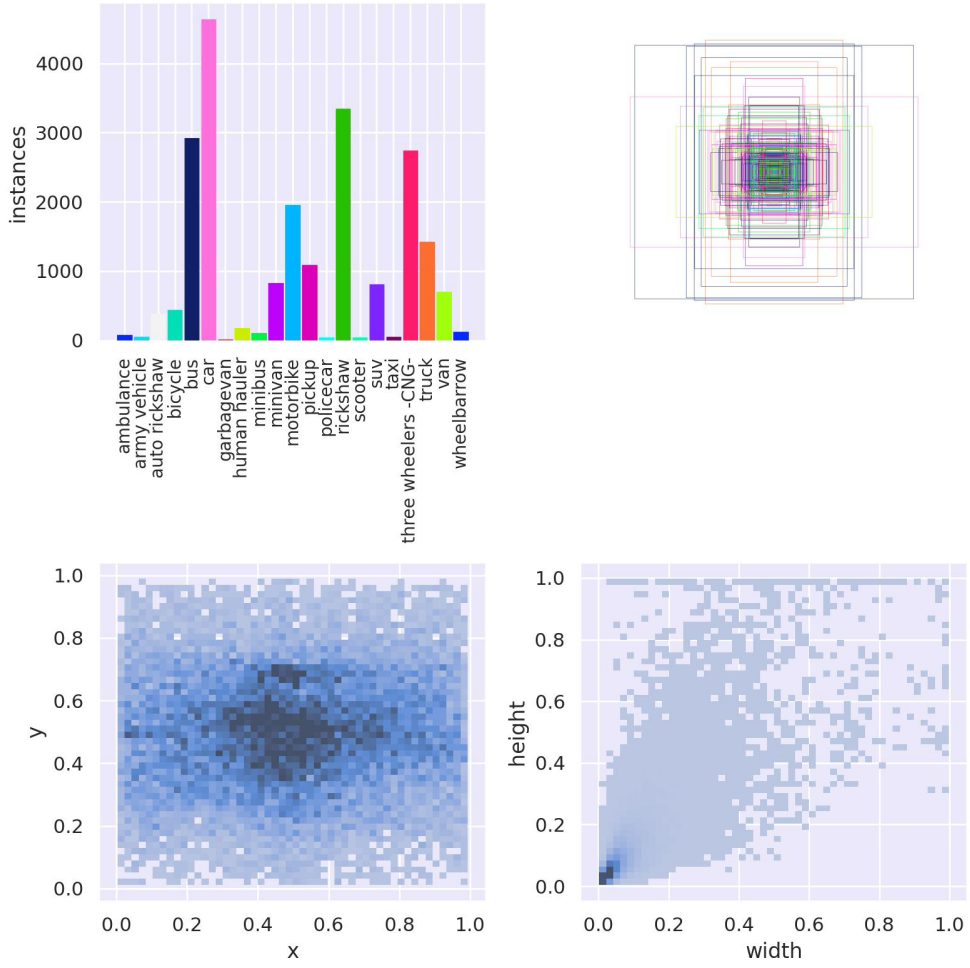


Figure-13: 21-Class Custom Trained Distribution Histogram

The primary challenge encountered was the limitation imposed by the computational resources available through Google Colab Pro. The inability to train the medium and large YOLOv8 models due to time and cost constraints represents a significant limitation of this project. These larger models, although potentially offering higher accuracy, were not feasible given the project's constraints. Consequently, the focus remained on optimizing the YOLOv8n model for real-time performance, albeit with the understanding that higher detection accuracy could have been achieved with more powerful models.

Here is the example of 21- class custom trained YOLOv8n traffic object detection output frame:

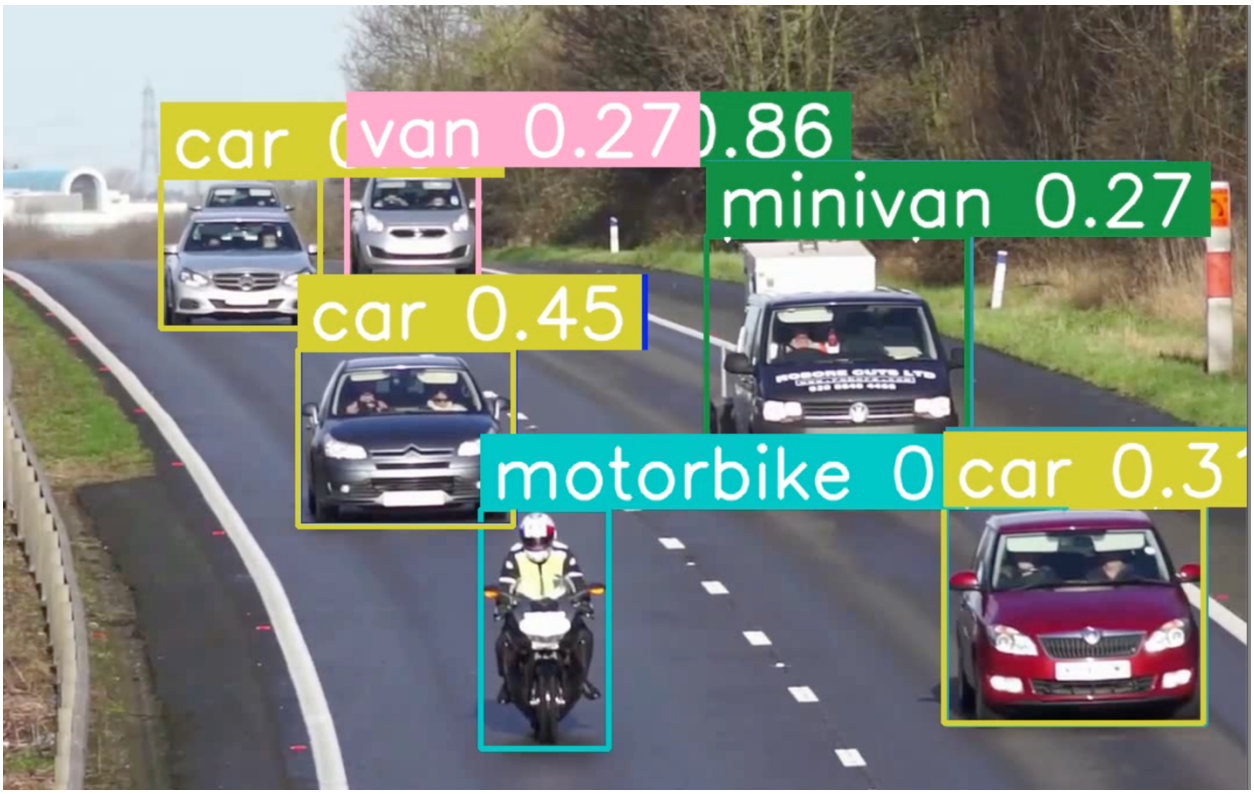


Figure-14: 21-Class Custom Trained Model Output Frame

4.2.5 YOLOv8 Nano Custom Trained on a Single Class Traffic Dataset

Another implemented custom model is the YOLOv8 Nano traffic single-class dataset. The dataset, obtained from a publicly available source, contains approximately 47 MB of data, focusing solely on a single class that includes vehicles such as cars, buses, bicycles, etc. The goal was to evaluate how well the YOLOv8n model could detect different types of vehicles when trained on a dataset that treats them as a single unified class, simplifying the detection process. The training was again conducted on the Google Colab Pro platform due to budgetary constraints, similar to the previous experiments. Given the limited computational resources, the model training was optimized for real-time performance. The custom training of the YOLOv8n model on the single-class dataset took approximately 8 hours. Since the dataset was smaller and consisted of only one class, the training time was shorter compared to the multi-class dataset training. (Kaggle, 2024b)

The F1-confidence curve whis is shown at Figure-15, illustrates the performance of the YOLOv8 Nano model trained on a single-class traffic vehicle dataset. As the confidence threshold increases, the F1 score

improves initially, peaking at 0.92 when the confidence is approximately 0.59, indicating the optimal balance between precision and recall. This demonstrates the model's ability to accurately detect vehicles while minimizing false positives at moderate confidence levels. However, as the confidence threshold increases further, the F1 score rapidly declines due to missed detections, suggesting that the model becomes too conservative in identifying vehicles at higher confidence levels. (Kaggle, 2024c)

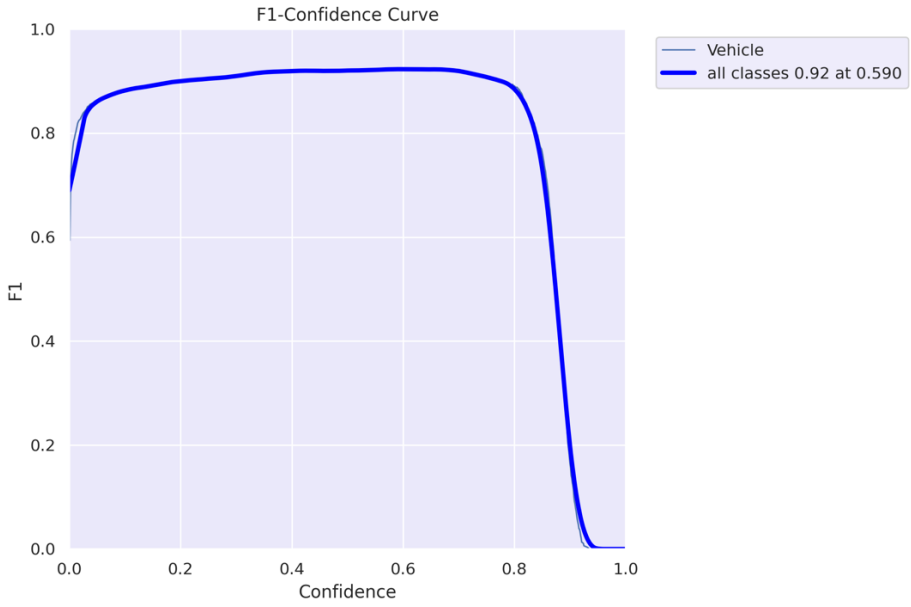


Figure-15: Single Class Trained Model F-1 Confidence Curve

Figure-16 presents the performance metrics of a custom-trained YOLOv8n model for single-class vehicle detection over 100 epochs. The top row displays the training metrics, including box loss, classification loss, and distributional focal loss, while the bottom row shows the corresponding validation metrics. Box loss measures the error in localizing objects, and its consistent decrease across training and validation indicates improved spatial accuracy in detecting vehicles. Classification loss, which reflects the model's ability to differentiate between vehicles and the background, shows a sharp reduction in the early epochs, stabilizing as the model learns with high confidence.

Distributional focal loss, which helps the model focus on harder samples for improved bounding box predictions, gradually decreases, indicating the model's refined capability in localizing objects. Precision and recall metrics show that, while precision fluctuates during early training, it stabilizes at around 0.85, representing the model's ability to reduce false positives. Recall, on the other hand, trends steadily upward, maintaining a value near 0.9, reflecting the model's sensitivity in detecting most vehicles.

The mAP50 and mAP50-95 metrics measure the mean Average Precision at different IoU thresholds. mAP50 approaches 0.95, while mAP50-95 stabilizes at approximately 0.75, indicating robust performance across various overlap thresholds. Overall, the model demonstrates strong generalization to the validation data, with consistent improvement in localization and classification accuracy, making it well-suited for real-time vehicle detection in traffic applications.

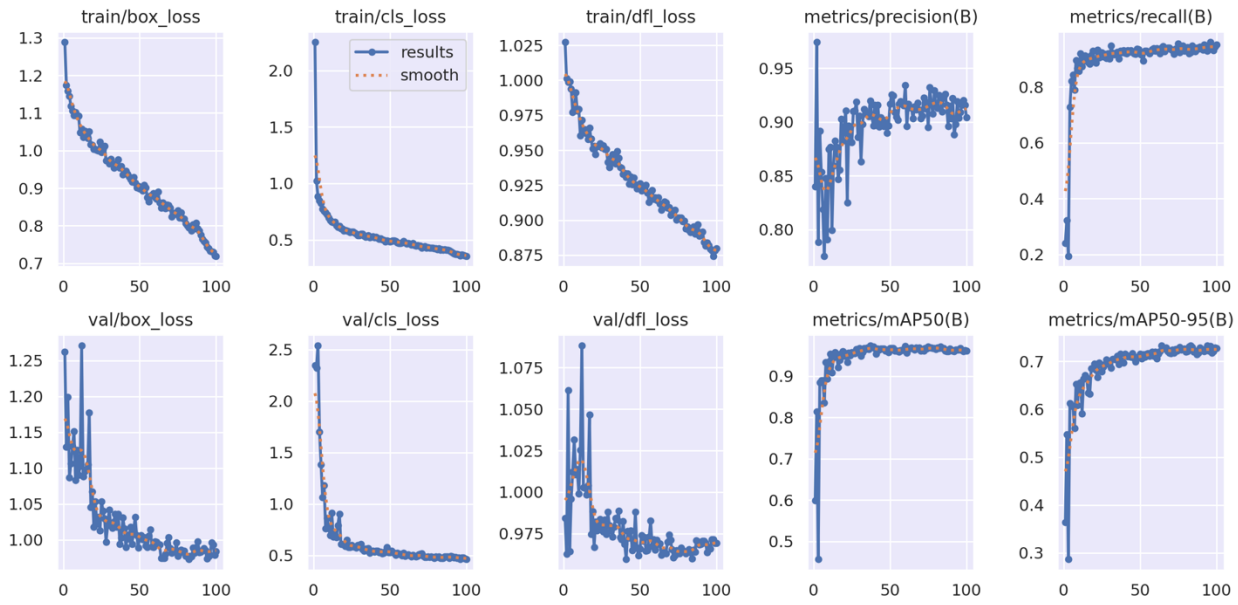


Figure-16: Single Class 100 Epocs Loss Algortim Values

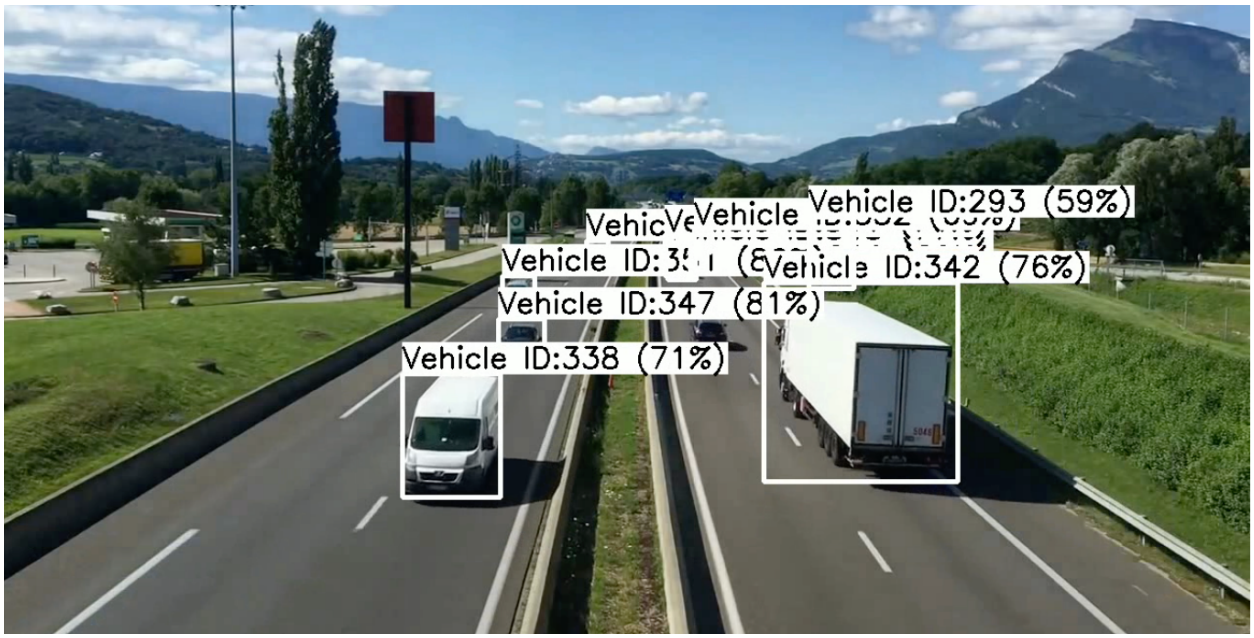


Figure – 17: Single Class Trained Model Output Frame

4.3 Web-Based Video Processing Application and Object Detection

A web-based video processing application was developed using the combination of, Intellij as an IDE, Python-Flask framework and HTML as front-end. Those interfeaces are used to enable traffic object detection and density estimation. The system integrates an intuitive interface that allows users to upload videos, select among several YOLOv8 models which are trained before and integrated in the system, visualize the results of the video processing directly on the platform. The web application is composed of three main components: upload.html, processed.html, and the back-end app.py, all of which have been designed to ensure efficient processing and user-friendly interaction. The full codebase of this project will be stored in the UoBD's GitLab repository.

The upload.html component serves as the primary front-end interface where users can select between multiple pre-trained and custom-trained YOLOv8 models (YOLOv8 Nano, Medium, Large, 21-class and single class custome trained models). The user uploads video files, which are processed to perform object detection and traffic analysis. The interface, implemented with HTML5 and Bootstrap, includes dynamic CSS styling for visual appeal, ensuring responsiveness and ease of interaction across different device types. Once a video is uploaded, the system processes the file using the selected model, leveraging the Flask framework to manage back-end operations and file handling. The design prioritizes ease of use and flexibility, allowing users to switch between models depending on their desired detection accuracy and inference speed.

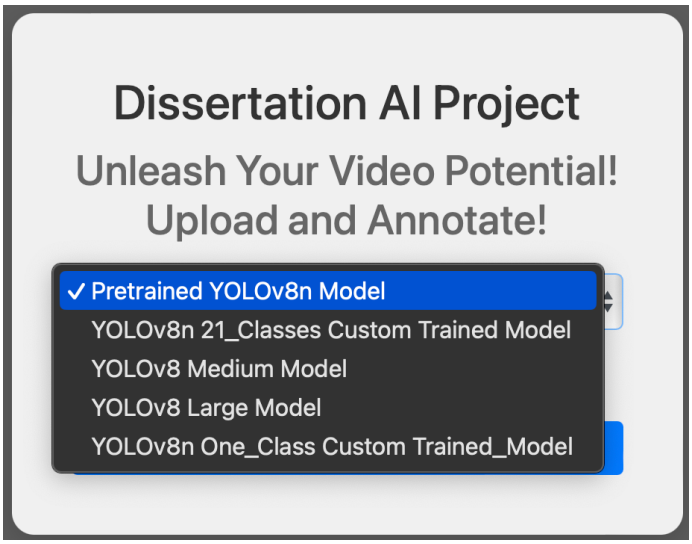


Figure-18: Model Selection Menu

The dataset used for custom training and evaluating the YOLOv8n custom modesl were sourced from a publicly available traffic-specific dataset on Kaggle, containing approximately 185MB of data across 21 different traffic-related object classes. This dataset was further supplemented by a custom-collected dataset of 47MB, focusing on a single unified class for simpler detection scenarios. The data includes diverse traffic scenes such as highways, intersections, and urban roads, annotated with bounding boxes for vehicles, motorbikes, buses, and other objects. The preprocessing of the dataset involved resizing all video frames to 640x640 pixels and manually annotating any missing or misclassified objects to ensure high-quality training input.

The results of the video analysis are displayed through processed.html, where a video player presents the processed footage with real-time object detection overlays. Additionally, a traffic density report, which includes metrics such as total vehicle count, average speed, and categorized vehicle types, is dynamically generated and presented to the user. A log section records the processing stages, providing transparency into the analysis pipeline and helping users track the system's operations. The front-end's interactive elements, built using Bootstrap, allow users to toggle through detailed traffic reports and reprocess videos if needed. This comprehensive feedback mechanism supports both casual users and professionals who require precise traffic analytics.

All the back-end logic, including video processing and model integration, is encapsulated in the app.py file. Flask routes are used to handle file uploads, trigger model inference using pre-trained or custom-trained YOLOv8 models, and dynamically generate the traffic reports. The processed results are then passed to the front-end for display. This architecture balances the need for computational efficiency with user accessibility, offering a scalable solution for real-time traffic monitoring and density estimation. Furthermore, the complete source code and documentation for this project will be made available on the university’s GitLab repository, ensuring long-term availability for future research, collaborations, and potential enhancements by the academic community.

In summary, the implementation of Object Detection System visualizes the proper implementation of the YOLOv8 Nano model in traffic object detection to point out its balanced feature between computational efficiency and detection accuracy. Trained only with custom and pre-trained traffic datasets, it shows a good performance in detecting a variety of vehicle classes by considering both resource constraints and imbalanced classes. The use of key evaluation metrics IoU and mAP showed that the model had a high degree of accuracy, hence suitable for real-time applications. More importantly, a web-based video processing application enhanced system usability, enabling the monitoring of traffic in real time and estimating its density. Although the system was found to perform well in real-world scenarios, limitations included lower accuracy of the classifier for less frequent vehicle classes, and the use of a smaller model version was used since resource-limited applications were being targeted, hence pointing to scalability and further refinement of the detection accuracy as areas of future work.

5 Design and Implementation of the Traffic Density Estimation System

Traffic density estimation is the second main part of the project. The system implements a traffic density estimation algorithm as part of object detection and video processing pipelines. The system is able to use two different type of custom trained YOLOv8n models or medium,large models for object detection and integrates a multi-step process to estimate traffic density based on the number of vehicles detected within a given time frame. The following outlines the scientific methodology and algorithmic design that underpins the traffic density estimation process in its code.

The first critical component of the system is object detection, which is accomplished using a pre-trained or custom-trained YOLOv8 model. This model processes each frame of the video, identifying and labeling various vehicles (e.g., cars, buses, motorbikes) using bounding boxes and confidence scores. The object detection outputs bounding boxes for each detected vehicle along with a class label and confidence score. The detections from YOLOv8 are then passed into a tracking algorithm based on the SORT (Simple Online and Realtime Tracking) method, which assigns unique IDs to objects (vehicles) across frames. This is essential for tracking the same vehicle over multiple frames, allowing for calculations related to movement and speed. But the speed calculation is a estimate value that depends on the frame per second value that will be explained in further paragraphs.

The primary metric for traffic density is the total number of unique vehicles detected in each time segment. The code processes the video in 10-second intervals (based on the frame rate), where each interval represents a "segment." For each segment, the system maintains a set of unique object IDs (unique_ids), which represent the distinct vehicles detected within that segment. The size of this set, i.e., the number of unique IDs, serves as a proxy for the number of vehicles in that time segment. This count is then directly used as the measure of traffic density.

Segmentation: The video is divided into temporal segments, typically 10 seconds long, by calculating the number of frames per segment using the frame rate (fps). This segmentation is crucial to localizing traffic conditions within specific periods, providing a more granular view of traffic flow.

Density Estimation: At the end of each segment, the code computes the total number of vehicles in the segment (total_vehicles) by counting the unique object IDs assigned to detected vehicles by the tracking algorithm. This total is stored in the traffic_density variable, which represents the traffic density for that segment. Additionally, the total number of vehicles detected in the segment is considered directly as the traffic density level, avoiding the need for normalizing by the number of frames in each segment.

The system is designed to estimate vehicle speed, a critical parameter for analyzing traffic conditions. The speed estimation relies on tracking the movement of vehicles over time by calculating the distance they travel between consecutive frames, using the center of the bounding box for each tracked object. This distance, initially measured in pixels, is converted to real-world meters through a pixel-to-meter conversion factor. This factor is derived from the real-world dimensions of a reference object, such as a car, whose size is manually measured both in meters and pixels from the video. In the current implementation, the real-world length of the reference object (a car) is set at 5 meters, and its pixel length is recorded as 40 pixels, giving a conversion factor of 0.125 meters per pixel. This method is adaptable and provides the flexibility to adjust critical parameters such as frames-per-segment, which is calculated as "frames_per_segment" = int(fps * 10) to define 10-second segments based on the frame rate.

Furthermore, variables like “reference_object_real_length” and “reference_object_pixel_length” can be adjusted to fit different scenarios. As an example, the system allows the administrator to modify the reference object dimensions depending on the camera’s perspective which comes from height, or zoom level. (That feature is not implemented for user, yet it can be adapted to UX part of the project) It can influence pixel measurements. Pixel adaptation is essential, as each traffic monitoring setup may have distinct characteristics depending on the camera configuration and environmental factors such as the type of road (highway, urban, or rural areas). By allowing for dynamic adjustments to that pixel parameters, the system can be fine-tuned for varying conditions, making it an ideal solution for a flexible and adaptive traffic estimation framework. This capability ensures that the system remains robust and accurate across different traffic surveillance environments, optimizing detection and traffic flow analysis based on the specific characteristics of the deployed cameras and surrounding conditions.

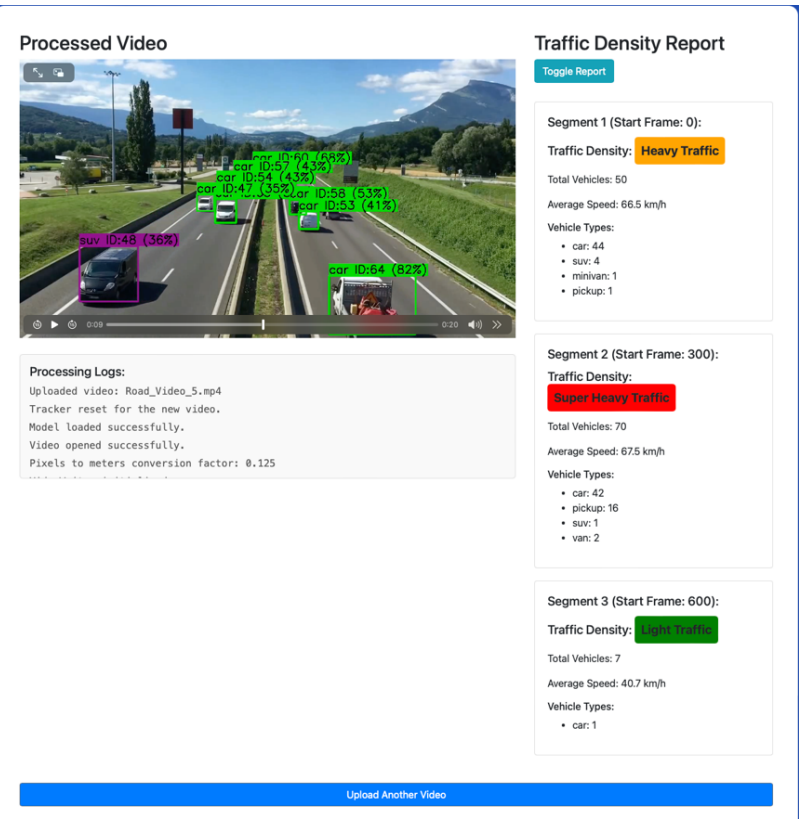
Distance Calculation: The Euclidean distance between the current and previous positions of the vehicle's center is calculated in pixel space. This pixel distance is converted into real-world meters using the formula:

$$\text{distance in meters} = \text{distance in pixels} \times \frac{\text{real-world length of reference object}}{\text{pixel length of reference object}}$$

In this case, the real-world length of the reference object (a car) is set to 5 meters, and its pixel length is manually measured as 40 pixels, yielding a conversion factor of 0.125 meters per pixel.

Speed Calculation: Once the distance traveled by a vehicle is known, its speed can be calculated by dividing the distance by the time elapsed between two following frames. This yields the speed in meters per second (m/s), which is then converted into kilometers per hour (km/h). The calculated speeds are stored in each segment’s metrics and averaged to provide an overall speed estimate for the segment.

To provide a more understandable measure of traffic conditions, the code includes a function called: “calculate_traffic_density_level”, which classifies traffic density into categories such as “Light Traffic”, "Moderate Traffic", “Heavy Traffic”, and “Super Heavy Traffic”. These categories are determine by thresholds based on the total number of vehicles in each 10-second segment. In example, segments which are more than 50 vehicles are classified as “Super Heavy Traffic” while those are fewer than 15 vehicles are classified as “Light Traffic”. The traffic density category is then color-coded for display in the report, using colors such as red for “Super Heavy Traffic” and green for “Light Traffic”. Please see in below, two different custom trained traffic density reports,logs and traffic analysis:



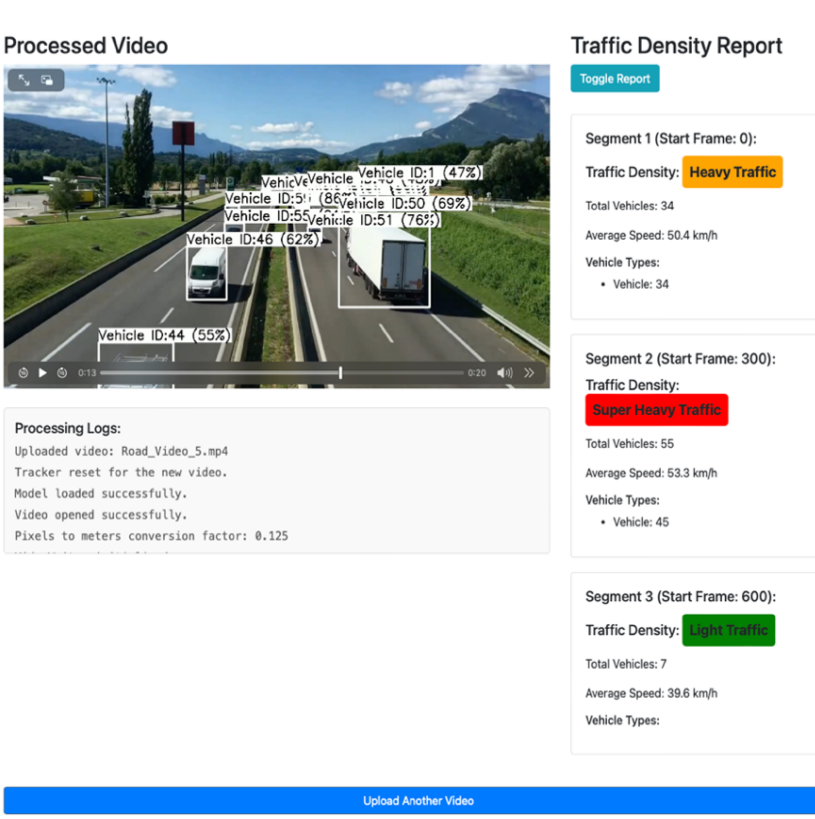


Figure-19 and 20:
21-Classes and Single Class Custom Trained Traffic Density Reports

In conclusion, the traffic density estimation system represents a robust and adaptable solution for real-time traffic analysis and monitoring. By leveraging advanced object detection models like custom-trained YOLOv8 and integrating them with the SORT tracking algorithm, the system efficiently identifies and tracks vehicles across video frames. The segmentation of video into 10-second intervals allows for granular analysis of traffic flow, providing a dynamic view of traffic conditions over time.(See Figure-19 and Figure-20)

Flexibility in design will be developed by accommodating parameters in a tunable manner, which includes but is not limited to those indicating frame rate, object reference dimension, pixel-to-meter conversion factors. This adaptability will allow the system to be fine-tuned for many different camera configurations and environmental conditions, further enhancing the applicability of the system on diverse traffic surveillance setups. Precise conversions of pixel measurements into real-world distances allow for accurate estimations of speed-a very important dimension of traffic analysis.

It measures traffic density in intuitively understandable levels such as "Light Traffic," "Moderate Traffic," "Heavy Traffic," and "Super Heavy Traffic." Color coding enhances the visual comprehensions within these reports for quick assessment of the conditions of flow. In general, the integration of object detection, tracking, and dynamic parameter adjustment positions this system as the comprehensive tool in optimizing traffic flow analysis and contributing to improved traffic management strategies.

6 Areas of Development and Conclusion

6.1 Strengths of the Implemented System

The foremost advantage of the constructed system is attributed to its implementation of advanced object detection algorithms as a pre-trained model. In particular, the YOLOv8 model balances its rapidity and accuracy. By embedding this model with datasets specifically tailored for traffic, the system can accurately identify and categorize various traffic objects, such as cars, buses, and motorcycles, from real-time situations. Such functionality is critical to traffic management applications, as the needs for both rapidity and precision in detection are paramount. The capabilities of this system to process video streams near real-time while maintaining a high detection rate make it particularly suitable for smart city implementations and urban traffic monitoring contexts.

Another important benefit is the system's flexibility: Using pre-trained models and further training them on data representing traffic conditions allows the system to be flexible and scalable. Moreover, the integration of an intuitive web interface ensures ease of access even for users without a technical background. In addition, compatibility with mobile and fixed-position devices might contribute to easier adoption for municipal authorities and other stakeholders working to improve traffic flow.

The YOLOv8 architecture, with its ability to process large amounts of data efficiently, provides the system with the capacity to handle complex real-time traffic environments. This efficiency is further enhanced by the integration of a video processing pipeline that supports both pre-trained and custom-trained models. As such, the system is not only capable of detecting a wide range of vehicles but also estimating traffic density in real-time.

6.2 Limitations of the System

Despite the strengths mentioned above, the current system has limitations that present challenges for large-scale deployments. One of the primary limitations is related to the computational requirements of training larger models, such as YOLOv8m or YOLOv8l. As discussed in the dissertation, the system primarily relies on the YOLOv8 Nano model due to the limited resources available on platforms like Google Colab Pro. While the YOLOv8 Nano model is highly efficient for real-time applications, its reduced complexity leads to lower detection accuracy compared to larger models. This trade-off between speed and accuracy poses a significant limitation, particularly when the system is deployed in environments with diverse traffic conditions, such as highways or rural roads, where detection of smaller or occluded objects is critical.

Another limitation lies in the system's generalization capabilities. The system's performance is dependent on the quality of the dataset used for training. Although one of the custom datasets used for training the model included 21 traffic-related object classes, it may not adequately cover all possible traffic scenarios, particularly those involving less common vehicles or challenging weather conditions. This limitation can reduce the model's accuracy when deployed in regions with different traffic characteristics than those represented in the training data.

6.3 Future Work and Potential Development Areas

6.3.1 Improving Model Accuracy through Advanced Training Techniques

A potential area for further research could be tuning the precision of the object detection model. While the current framework uses the YOLOv8 Nano model, future versions might investigate training for larger models such as YOLOv8m or YOLOv8l to enable more accurate detection. Limitations due to the high computational cost of training these models using services like Google Colab Pro can be overcome by accessing more advanced hardware, such as a GPU, or employing cloud-based services that support machine learning. In addition, larger model variations could be tested by leveraging transfer learning techniques, which would reduce the training time required when fine-tuning on the prebuilt custom traffic dataset.

6.3.2 Integrating Multiple Data Sources for Enhanced Traffic Monitoring

Future researches can focus on other sources of data input, besides the detection of visual objects, to further improve the system's accuracy and usefulness. For instance, the system could include live data generated by CAVs as part of its data source, as indicated in one of the reviewed papers. A system that combines data from camera feeds with CAV data could provide more comprehensive estimates of traffic density, particularly in scenarios where camera-based detection is limited, such as heavy fog or nighttime conditions. This would indeed present a hybrid approach that offers a more robust solution for traffic management by considering both the visual and non-visual streams.

6.3.3 Exploring Advanced Tracking and Post-Processing Techniques

Currently, vehicle tracking between frames is performed using a Simple Online and Realtime Tracking algorithm. A future project may replace this with any number of more complex tracking methods, including DeepSORT or temporal information using RNNs. Going forward to such an implementation with these advanced algorithms will lead to more stability in vehicle tracking but more accurately, especially in the case of high-density traffic when objects of interest often occlude others. Smoothing algorithms could potentially further improve the uniformity of bounding box positioning and reduce flickering in the output of the detection.

6.3.4 Developing a More Comprehensive User Interface

The current web-based interface offers basic functionality for uploading and processing videos, but future development could focus on creating a more interactive and customizable user experience. For instance, the system could allow users to adjust detection thresholds, select different models based on their needs, and export detailed traffic reports in various formats (e.g., CSV, PDF). Additionally, the integration of real-time alerts and notifications based on traffic density levels could further enhance the system's utility for traffic management authorities.

6.4 Conclusion

In this regard, object detection using YOLOv8 has significant potential for real-time traffic monitoring, detection, action taking measures and density estimation status. The model selection was done with limitations of focus on the YOLOv8 Nano version to balance computational efficiency with the needed detection accuracy in being computationally feasible in resource-constrained environments. This was improved by training models with a traffic-specific dataset in real-world conditions, but class imbalances, reduced detection accuracy for less common object classes, and a number of computational constraints when training larger models hindered the system from achieving its full potential. Despite the above-mentioned limitation, the system showed a reliable performance in the detection, especially for real-time traffic situations. It also provided users with an interactive platform to monitor and analyze traffic conditions through its web-based interface. Model integration can further be enhanced, and more significant improvements can be achieved with more data, for better generalization, to propose larger-scale deployments in smart city infrastructure. Besides, there is still much room for improvement in the tracking algorithms and for new modules development to incorporate data sources other than visual. In summary, this project sets a very firm base in the detection of objects in traffic and in density estimation but at the same time presents various avenues where future research and development could be done.

7 References

Zou Z., Chen.K, Shi Z., Guo Y. and Ye J., 2022. Object Detection in 20 Years: A Survey. *International Journal of Computer Vision*, 130(6), pp.1054-1100. doi: 10.1007/s11263-022-01528-0.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.779-788. doi: 10.1109/CVPR.2016.91.

Ren S., He K. , Girshick R., and Sun J. , 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), pp.1137-1149. doi: 10.1109/TPAMI.2016.2577031.

Bochkovskiy A., Wang C.Y. , and Liao H.Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934*. Available at: https://arxiv.org/abs/2004.10934

Zhang, X., Zhang, W. and Zheng, X., 2023. Real-time Object Detection Method Based on Improved YOLOv4-tiny. *Journal of Intelligent & Fuzzy Systems*, 44(3), pp.1515-1527. doi: 10.3233/JIFS-221327.

Daisik N. , Lavanya R. , Jayakrishnan R., Yang I. , and Hoon W. J., 2022. A Deep Learning Approach for Estimating Traffic Density Using Data Obtained from Connected and Autonomous Probes. *IEEE Transactions on Intelligent Transportation Systems*, 23(8), pp.11825-11838. doi: 10.1109/TITS.2021.3087569.

Sahasranand K.R. ,and Tyagi H.., 2020. Communication Complexity of Distributed High Dimensional Correlation Testing. *IEEE Transactions on Information Theory*, 66(8), pp.5169-5184. doi: 10.1109/TIT.2020.2996814.

Mittal U., and Chawla P. , 2021. Vehicle Detection and Traffic Density Estimation Using Ensemble of Deep Learning Models. *International Journal of Advanced Computer Science and Applications*, 12(3), pp.350-357. doi: 10.14569/IJACSA.2021.0120346.

Ultralytics, 2024a. YOLOv8 models. Available at: https://docs.ultralytics.com/datasets/detect/coco/

Ultralytics, 2024b. YOLOv8 models. Available at: https://docs.ultralytics.com/models/

Kaggle, 2024a. Kaggle open source datasets. Available at: https://www.kaggle.com/datasets/awsaf49/coco-2017-dataset

Kaggle, 2024b. Kaggle open source datasets. Avaible at: https://www.kaggle.com/code/farzadnekouei/real-time-traffic-density-estimation-with-yolov8/input

Kaggle, 2024c. Kaggle open source datasets. Avaible at: https://www.kaggle.com/code/ishanpurohit/yolov8-vehicle/input

8 Appendix

8.1 Git Repository

Erbak, E. (2023) *Object Detection FYP v1.7* [Software]. Available at: https://git.cs.bham.ac.uk/projects-2023-24/exe252
Erbak, E. (2023) *Object Detection FYP v1.7* [Software]. Available at: https://github.com/ermo7/TrafficObjectDetection

8.2 System Architecture for Traffic Object Detection and Density Estimation

8.2.1 Frontend Interface Design

The system's user interface serves as the point of interaction, providing users with an intuitive way to upload videos and retrieve processed results. This interface is divided into two main pages:

Upload Page (upload.html): The user can upload traffic video data and select a YOLO model for processing. The page is designed using HTML5 and Bootstrap to ensure responsive behavior across devices. Users are presented with multiple model options.The use of a clean design with form validation ensures ease of use and clarity in user actions. The uploaded video is then sent to the backend for processing via a secure POST request.

Processed Page (processed.html): After the video has been processed, the results are displayed on this page. The core of this page is the video playback component, where the user can view the annotated output, which includes bounding boxes, object classifications, and traffic density information. Additionally, a detailed *Traffic Density Report* is generated, showing vehicle counts, average speed, and vehicle types for different segments of the video. The system logs the object detection process for each frame and presents it in a *Log Section*, aiding in debugging and transparency of the detection process. This logging capability is essential for detailed performance evaluation and identifying edge cases where the model might struggle.

8.2.2 Backend Processing Workflow

Model Selection and Deployment: Upon receiving the video upload request, the backend identifies the selected YOLO model. Each model—YOLOv8n (nano), YOLOv8m (medium), YOLOv8l (large), or custom-trained models—is optimized for specific scenarios, including performance, computational cost, and accuracy. The backend is implemented in Python, leveraging PyTorch for model execution. The flexibility to switch between pre-trained and custom-trained models allows for comparative analysis and experimentation across different traffic datasets.

Frame-by-Frame Processing: The uploaded video is processed in a frame-by-frame manner. Each frame is passed through the selected YOLO model, which generates bounding boxes, class probabilities, and confidence scores for detected objects. These outputs are filtered based on a confidence threshold, ensuring that only high-confidence detections are retained.

Traffic Density Estimation: In addition to object detection, the system computes traffic density by dividing the video into segments. The traffic density is categorized based on the number of detected vehicles and their spatial distribution in each frame. The system can handle various traffic environments (e.g., highways, urban roads) and output real-time traffic density information for traffic management applications. Each segment's traffic density is visualized in the user interface, using color codes to represent different levels of congestion.

Post-Processing and Data Aggregation: The system aggregates the detection results from all frames and calculates the average speed of vehicles, the total vehicle count, and other relevant traffic metrics. These

results are then presented in a report format, highlighting key traffic parameters like the number of detected vehicles, traffic density levels, and the types of vehicles (e.g., cars, trucks, buses) detected.

These metrics help quantify the traffic situation, making the system highly relevant for smart city applications and autonomous driving research.

Model Selection and Optimization

YOLOv8, the latest iteration of the You Only Look Once (YOLO) object detection model, was chosen due to its balance of speed and accuracy, particularly for real-time traffic detection applications:

YOLOv8n (Nano): This lightweight version is highly optimized for environments with constrained computational resources, such as edge devices or cloud-based systems with limited GPU access (e.g., Google Colab). Its lower computational demand makes it ideal for real-time applications but with some limitations in detecting smaller objects.

YOLOv8m (Medium) and YOLOv8l (Large): These models are more computationally expensive but offer higher accuracy, making them suitable for environments where performance is more critical than computational cost. They are used for comparative analysis, providing insights into how model complexity influences detection accuracy and speed.

Custom YOLOv8 Models: Custom versions of YOLOv8 were trained on a traffic-specific dataset containing 21 vehicle classes and one class.

Post-Processing and Visualization

Bounding Box and Label Rendering: For each detected object, the bounding box and its label (e.g., "car", "bus") are rendered onto the video frames. This step ensures that the user can visually verify the performance of the detection model.

Traffic Density Report: The processed video is segmented, and each segment's traffic density is calculated and displayed. The report is interactive, allowing users to toggle between segments to analyze traffic conditions throughout the video. Each segment is color-coded based on traffic density, providing an intuitive view of congestion levels.

Logs for Transparency: Detailed logs of the processing steps are maintained and displayed in the user interface. These logs include information on the number of frames processed, detected objects, and any anomalies encountered during processing.

Integration with Smart Traffic Systems

This system is designed with scalability and flexibility in mind, making it suitable for integration into larger smart traffic monitoring infrastructures. By processing real-time video feeds, it can provide actionable data for traffic management systems, including vehicle counts, congestion analysis, and traffic density estimation. The modular nature of the system allows for future expansion, such as integrating additional data sources like LIDAR or IoT-based traffic sensors.

8.3 Test Plan

Test Execution Results		
Test Case ID	**Description**	**Status**
TC_UI_01	Validate video upload and model selection	Pass
TC_YOLO_01	Ensure correct YOLO model is applied	Pass
TC_OD_01	Validate vehicle detection and bounding boxes	Pass
TC_TD_01	Validate traffic density calculation	Pass
TC_RES_01	Validate results display and log correctness	Pass