

Investigating and Comparing YOLO Models for Roadside Object Detection

Orion Ghai¹

Received August 20, 2025

Accepted December 19, 2025

Electronic access January 15, 2026

The problem of roadside object detection is essential for ensuring vehicle safety and preventing life-altering car accidents. A potential solution to this problem is the utilization of deep learning-based object detection models, such as YOLO, that can detect and classify objects. In this paper, we investigate the performance of six different YOLO models for roadside object detection: YOLOv7, YOLOv8, YOLOv9, YOLOv10, YOLOv11, and YOLOv12. We analyzed the performance of these models for detecting and labeling common objects found on roads. Furthermore, we investigated the augmentation techniques mixup and cutmix for improving the performance of these models. Our investigation revealed that YOLOv9 is the best model, as it achieved a precision of 84.8%, a recall of 84.8%, and an F1 score of 84.8% on the KITTI dataset. Our experimentation with different augmentation schemes demonstrated that implementing a mixup with a mixing coefficient of 0.9 further improves the model's precision by 4.1% and its recall by 1.9%, resulting in an F1 score increase of 3.0%. Similarly, implementing a cutmix at 0.1 increases the model's precision by 5.3% and its recall by 0.7%, resulting in a F1 score increase of 2.9%. Overall, this work provides several recommendations for training object detection models for roadside object detection.

Keywords: Object Detection, Object Classification, YOLO, Bounding box, Augmentation Schemes, mixup, cutmix

Introduction

As more vehicles and drivers have gotten on the road, there has been a sharp increase in the number of fatal road accidents in the past few years. According to the World Health Organization, over 1.2 million people are killed due to car accidents, and around 20 million others suffer life-altering injuries¹. The consequences of these accidents go beyond the immediate fatalities because they place a strain on healthcare systems and emergency services. As the number of traffic accidents continues to grow, it becomes crucial to identify their main causes and improve the methods of preventing them. In a study done by the U.S. Department of Transportation, it was found that over 94% of vehicle accidents are caused by human error, including factors like distraction, impaired driving, and decision-making mistakes².

One of the most effective ways to address this issue is to introduce additional guidance via the implementation of autonomous safety systems on vehicles. To apply these safety features, one must solve the problem of road object detection and identification. Object detection and identification algorithms on vehicles can warn drivers of oncoming traffic or the possibility of a pedestrian or cyclist crossing in their path. These features can also prevent drivers from getting too close to another car and help them keep track of the road in unlit

locations.

Many computer vision algorithms have been developed for this purpose. Template matching is a basic approach to object detection, where a template is used to describe an object, and then objects in captured images are compared to the template to check if they match. Feature extraction and logistic regression classification algorithms are another approach to object detection and classification. A feature detector describes an object's unique features, like edges and textures³. A trained classifier is then used to classify the object. While these methods are effective in most object detection applications, they often struggle when completing more complex tasks.

One method that has recently shown excellent results in computer vision applications, including object detection and classification, is Convolutional Neural Networks (CNN). CNN's ability to handle complex and non-linear patterns makes it a prime candidate for being used in autonomous safety mechanisms⁴.

An effective application of CNNs in object detection is the development of the "You Only Look Once" (YOLO) algorithm. In this paper, we examined the performance of different variants of YOLO methods on the popular KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) dataset⁵ comprising different roadside objects. The dataset contains a collection of images of real-world road scenarios and LIDAR (Light Detection and Ranging) data, along with

¹ Los Gatos High School, California, United States of America

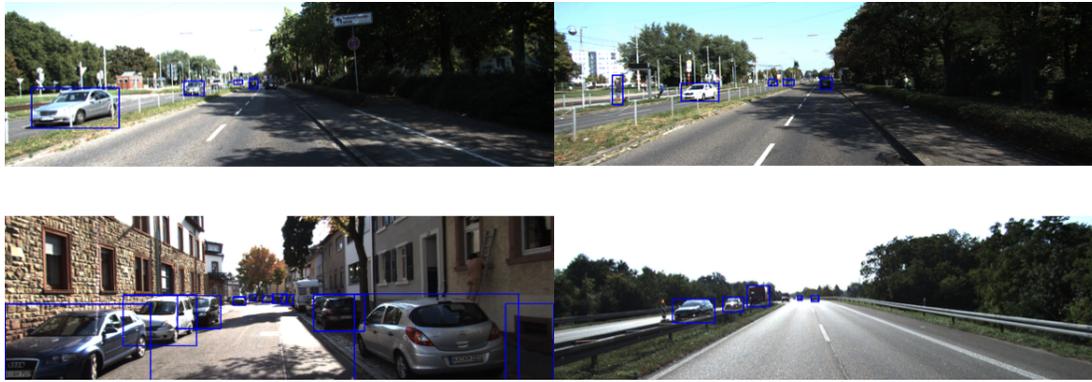


Fig. 1 Sample Images from Datasets.

nine different labels for the images. The different YOLO models used are YOLO v7, v8, v9, v10, v11, and v12. We have done comprehensive testing and tweaking of the models to maximize their effectiveness. A significant way of improving these models is by utilizing data augmentation. Hence, this work aims to compare how different YOLO models perform for object identification and classification tasks, and simultaneously determine the effect of mixup and cutmix-based augmentation schemes on each model.

The rest of this paper is organized as follows: first, related work relevant to this study is analyzed. Next, the dataset and methodologies used are described. This is followed by a detailed discussion of the experiments and results. Finally, the paper concludes with a summary and key findings.

Related Work

Research has been completed in this field already. Yang et al.⁶ present a comprehensive survey of image data augmentation techniques for deep learning, offering a structured taxonomy that includes geometric transformations, photometric adjustments, kernel filtering, and mixing-based methods such as mixup and cutmix. Their work synthesises empirical findings across multiple computer vision tasks, including object detection, demonstrating how augmentation improves dataset diversity and model generalisation. While the study does not propose new augmentation algorithms, it provides an essential conceptual and empirical foundation for understanding the practical role of augmentation in mitigating overfitting and improving detection robustness. This review informs the present work's selection of mixup and cutmix as augmentation strategies for YOLO-based roadside object detection on the KITTI dataset.

Chen et al.⁷ propose a robust object detection framework for autonomous driving that integrates contrastive learning with semi-supervised co-training and reinforcement learning-

driven bounding box augmentation. Published in Security and Safety, their method leverages unlabeled data to enhance robustness against environmental perturbations such as Gaussian noise, rain, and fog. Their focus lies primarily on improving detection stability under adversarial and adverse weather conditions through semi-supervised learning strategies. In contrast, the present study adopts a fully supervised training paradigm and concentrates on evaluating the impact of systematic image-level augmentation (mixup and cutmix) on YOLO model performance, allowing for a controlled analysis of augmentation effects independent of semi-supervised learning dynamics.

Recent reviews of deep learning in autonomous driving systems highlight the central role of convolutional neural networks in real-time perception tasks, including object detection and decision-making⁸. These studies emphasise the importance of accurate and timely detection for Advanced Driver Assistance Systems, particularly in complex roadside environments. This broader context underscores the relevance of optimising detection architectures and training strategies to improve system reliability in safety-critical scenarios. The present work aligns with these objectives by focusing on empirical optimisation of YOLO-based detection performance through augmentation techniques tailored to roadside object detection.

Zhao et al.⁹ introduce RT-DETR, a real-time transformer-based detector that eliminates Non-Maximum Suppression through a hybrid encoder architecture. RT-DETR demonstrates competitive accuracy relative to leading YOLO models while achieving increased inference speed compared to prior transformer-based methods. Although transformer detectors represent a promising alternative to traditional CNN-based architectures, the present study maintains a focused scope by evaluating only YOLO variants. This decision enables a more rigorous and controlled comparison within a single architectural family, providing depth in performance analysis rather

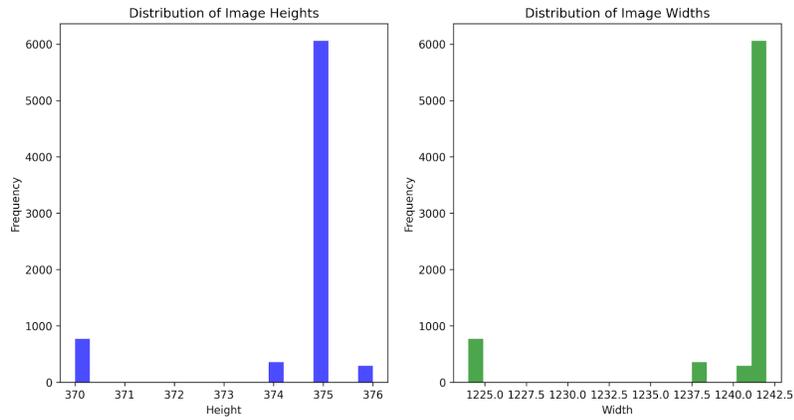


Fig. 2 Distribution of Image Heights and Widths.

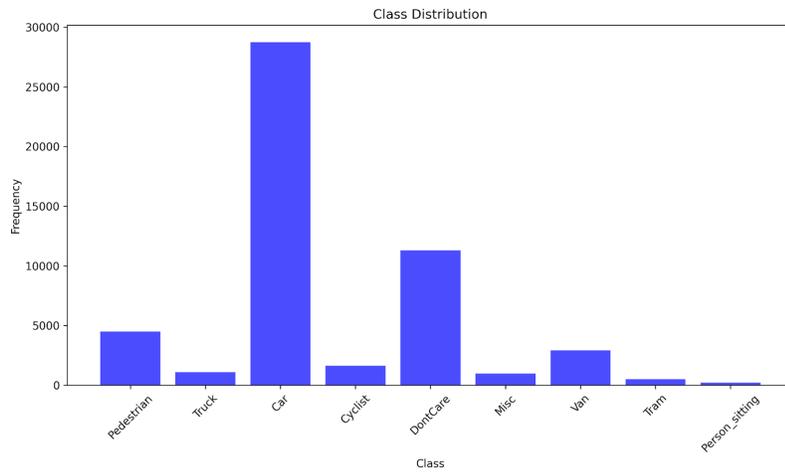


Fig. 3 Class Distribution.

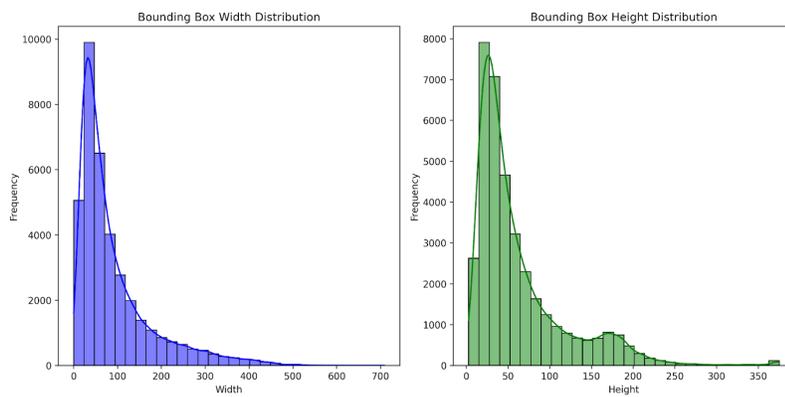


Fig. 4 Bounding Box Size Distribution.

than broad cross-paradigm comparison.

Vu et al.¹⁰ address the limitations of standard mixup in object detection through their lossmix framework, which interpolates loss values rather than ground truth labels to better accommodate the structured nature of detection tasks. Achieving state-of-the-art performance on PASCAL VOC and MS COCO, their work highlights the challenges of directly applying classification-oriented augmentation techniques to detection problems. While lossmix proposes an alternative formulation, the present study investigates conventional mixup and cutmix to systematically evaluate how standard augmentation strategies influence YOLO detector performance across different augmentation strengths in autonomous driving contexts.

Zhao et al.¹¹ propose X-Paste, an extension of copy-paste augmentation that employs text-to-image generation guided by CLIP to synthesise diverse training instances without manual annotation. This approach specifically targets data scarcity and class imbalance, improving detection performance for underrepresented categories. While X-Paste focuses on generating novel synthetic data, the present work explores how manipulating existing images through mixup and cutmix influences detection robustness, particularly for roadside-relevant classes such as cyclists and pedestrians.

Jia et al.¹² introduce an enhanced YOLOv5 architecture for autonomous driving applications, integrating structural reparameterisation, neural architecture search, and improved small object detection layers. Their model reports strong performance on KITTI, achieving high detection accuracy alongside real-time inference speeds. This work exemplifies how architectural refinement can significantly enhance detection effectiveness. In contrast, the present study explores performance gains achievable through training-time augmentation alone, allowing for comparison between architecture-driven and augmentation-driven performance improvements.

Similarly, Zhang et al.¹³ present MobileYOLO, a lightweight YOLOv4 variant designed for deployment on resource-constrained platforms. By reducing parameter count while maintaining competitive accuracy on KITTI, MobileYOLO highlights the trade-offs between efficiency and accuracy in real-world systems. The current study complements this line of research by examining whether augmentation strategies can further enhance detection accuracy without increasing computational overhead.

Wei et al.¹⁴ provide a comprehensive review of the YOLO family within autonomous driving contexts, tracing the algorithm's progression from early versions to the most recent iterations and summarising architectural improvements and deployment considerations. Their survey situates contemporary YOLO variants within a broader historical and technical framework. Building on this context, the present work conducts an empirical comparison of recent YOLO variants under uniform training conditions, with a particular focus on the in-

fluence of mixup and cutmix augmentation on roadside object detection performance using the KITTI benchmark.

Despite extensive research on architectural optimisation and emerging detection paradigms, limited work has systematically examined the interaction between augmentation strength and performance consistency across multiple YOLO variants in autonomous driving contexts. Existing studies tend to isolate either architectural innovation or augmentation technique evaluation, but rarely explore their combined influence under controlled experimental conditions. This study addresses this gap by conducting structured experiments on recent YOLO models, varying mixup and cutmix coefficients to quantify their effects on detection accuracy and robustness. The findings provide practical guidance for selecting augmentation strategies and parameter configurations in safety-critical roadside detection systems, where both precision and resilience to occlusion and environmental variation are essential.

Materials and Methods

Dataset and Pre-processing:

The systematic experiments run for this paper all utilize the KITTI dataset created by Andreas Geiger. The KITTI dataset was recorded from a moving car driving around the streets of Karlsruhe, Germany. The KITTI dataset was selected in this study because it provides a solid foundation for object detection models that need to function in real-world driving environments. The dataset contains images of real traffic situations one may encounter on a road, along with a mix of urban streets, highways, and residential areas. The images in the dataset have consistent image sizes and well-labeled annotations. All of these factors contribute to the KITTI dataset being the ideal dataset for testing the applicability and reliability of a model for roadside object detection. Sample images from the dataset are provided in Fig. 1. The dataset contains 7,481 images which were taken on daily sidewalks, streets, and roads. Image heights range from 370 to 376 pixels, with an average of 374.5 pixels per image. Image widths range from 1224 to 1242 pixels, with an average of 1239.9 pixels per image. The distributions of image heights and widths are depicted in Fig. 2.

There are 9 different classes of objects in the KITTI dataset, which are divided into these 9 labels: Pedestrian, Truck, Car, Cyclist, DontCare, Misc, Van, Person_sitting, and Tram. In total, there are 51,865 instances of objects. The distribution of classes is as follows: Car (28,742), DontCare (11,295), Pedestrian (4,487), Van (2,914), Cyclist (1,627), Truck (1,095), Misc (973), Tram (511), and Person_sitting (222). A visualization of the class distribution is provided in Fig.3. The dataset contains more instances of cars than of any other class because

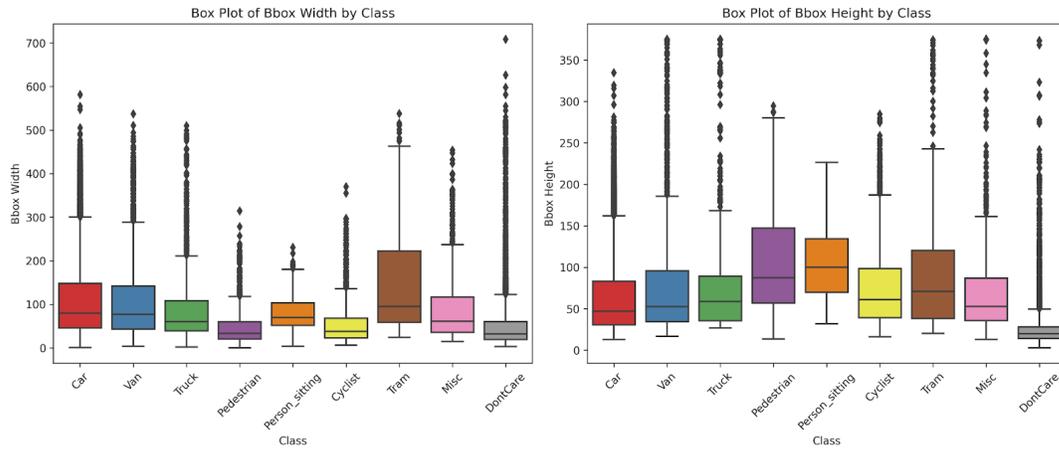


Fig. 5 Box Plots of Bounding Box Sizes by Class.

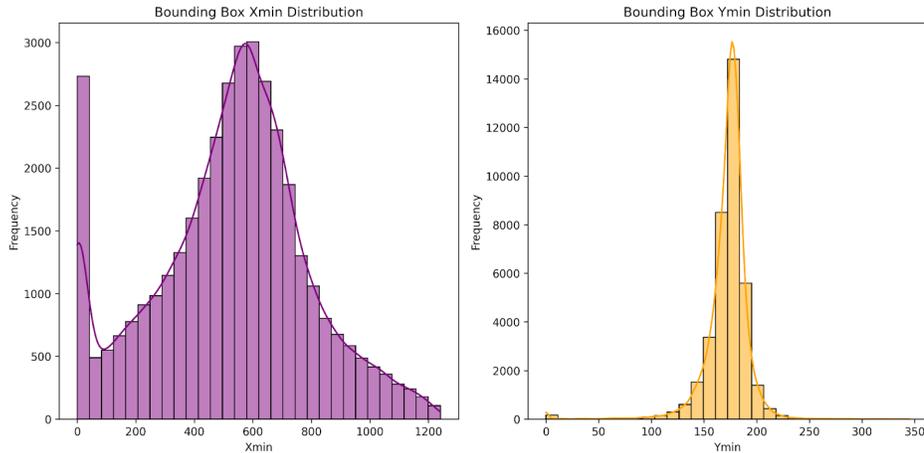


Fig. 6 Bounding Box Position Distribution.

cars are the most commonly found objects on roads. Other classes, like Tram or Person_sitting, have fewer instances because they are less frequently found when driving. The mean instances per class is about 5,763, while the standard deviation of instances per class is about 8,747. Each instance has a bounding box drawn around it. The majority of the bounding boxes have widths running from 25 to 150 pixels, with an average width of 91.1 pixels and a median width of 59.7 pixels. The heights of the bounding boxes primarily range from 10 to 75 pixels, with an average height of 63.0 pixels and a median height of 42.5 pixels. Fig.4 provides a visualisation of the bounding box size distribution for the dataset. The bounding boxes are primarily small because most objects in the images are being captured from a large distance. Fig.5 displays box plots for bounding box sizes by class. Trams have the largest median width, while pedestrians have the lowest. Pedestrians,

however, have the highest median image height, while cars have the lowest.

A bounding box's xmin represents the distance from its left-most point to the left edge of the image, and its ymin represents the distance from its bottom-most point to the bottom edge of the image. Most of the bounding boxes' xmin are concentrated between 0 to 40 pixels or 500 to 600 pixels. This suggests the majority of the objects appear on the left side, representing oncoming traffic, or on the center-right side of the image, representing roadside objects. The bounding boxes' ymin are concentrated between 150 to 200 pixels. This suggests that objects are likely located in the center of the image, where the roadline is located. Fig.6 shows visualizations of the xmin and ymin distributions for the bounding boxes in the dataset, and Fig.7 displays a scatter plot for bounding box positions.

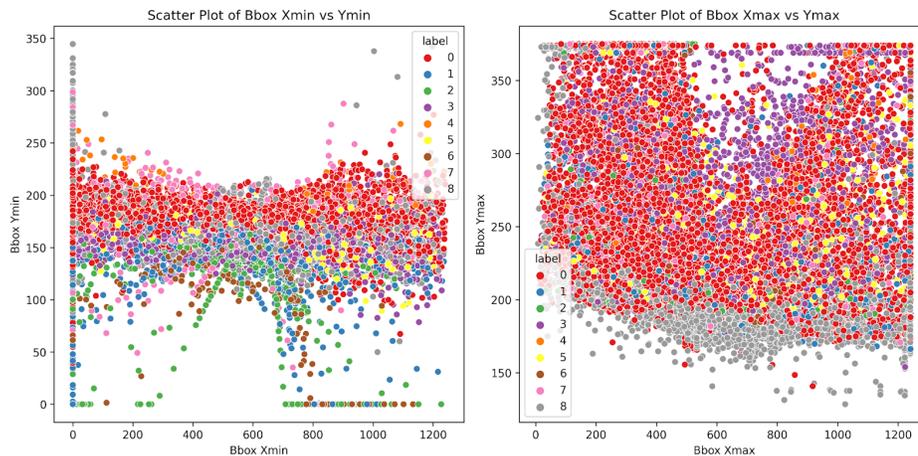


Fig. 7 Scatter Plot of Bounding Box Positions.

Fig.8 shows a correlation heatmap between the bounding boxes' width, height, xmin, ymin, xmax, and ymax values. The relationship between xmin and ymin and the relationship between xmax and ymax each have a strong negative correlation of -0.25 and -0.27, respectively. This suggests that objects on the sides of the image appear to be higher up in the image, while objects in the center of the image appear lower in the image. This pattern reflects how distant or elevated objects appear towards the edges and higher up.

YOLO Models

YOLO (You Only Look Once)¹⁵, proposed in 2015, is a state-of-the-art object detection algorithm that uses an end-to-end neural network to predict bounding boxes and class probabilities simultaneously. Instead of detecting possible regions of interest, YOLO performs all its predictions with connected layers, making it faster than other algorithms like R-CNN(Region-based Convolutional Neural Network), which identifies potential object regions in an image and then uses a neural network to classify what each region contains, or DPM(Deformable Part Models), which represent objects as collections of parts that can flex and move relative to each other while maintaining spatial relationships¹⁶.

YOLO takes images as inputs and uses a deep convolutional neural network to detect objects in the images. The architecture of the CNN model YOLO utilizes is shown in Fig. 9. YOLO divides the input image into a square grid. When an object is present in the image, each grid cell predicts a bounding box and produces a confidence score for the bounding boxes. Each grid cell can predict multiple bounding boxes, but during training, the algorithm chooses whichever bounding box overlaps most with the actual object location. As training progresses, the grid cells become better at predicting bounding

boxes¹⁷.

In this paper, we have implemented YOLO models v7, v8, v9, v10, v11, and v12. Descriptions for each YOLO model implemented are provided below:

YOLOv7

YOLOv7¹⁸, released in July 2022, is a single-stage real-time object detection system. YOLOv7 focuses on the optimization of the training process and introduces several key features and improvements over previous YOLO versions. It proposes a planned re-parameterized model, a technique that can be used in different types of neural network layers with the concept of gradient propagation path.

The model also introduces a new label assignment method called coarse-to-fine lead guided label assignment. This method starts with a rough decision about which network layer should detect each object, then gradually refines those assignments based on which layers performed the best. The model also proposes extended and compounded scaling methods for the real-time object detector¹⁹.

Compared to previous models, YOLOv7 reduces parameter count by 40% and computation by 50%, while achieving a faster inference speed and a higher detection accuracy.

YOLOv8

YOLOv8²⁰, released by Ultralytics in January 2023, builds on the advancements of previous yolo versions and introduces new features and optimizations. The model employs backbone and neck architectures. The backbone extracts features from raw images, and the neck combines and organizes these features. This allows the head to make final predictions on the

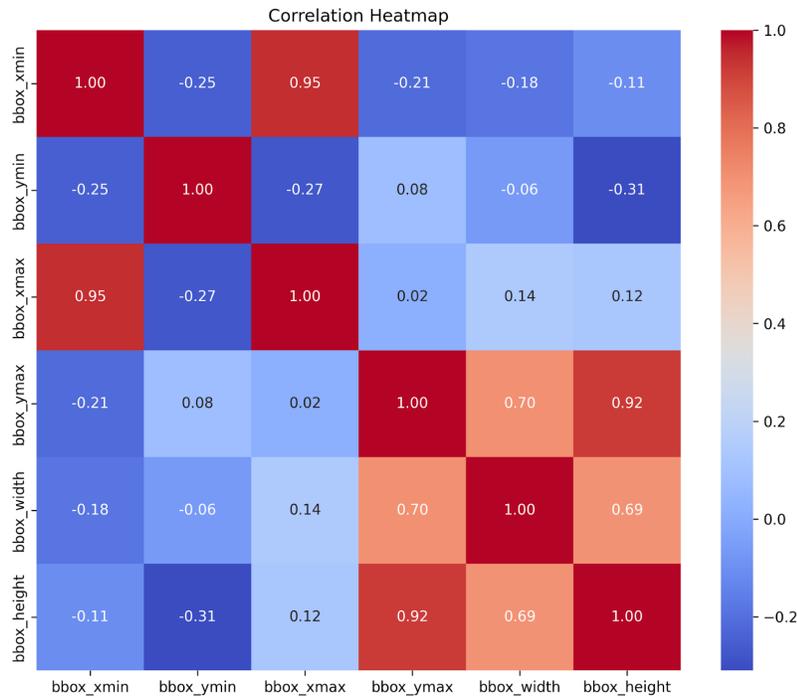


Fig. 8 Correlation Heatmap of Bounding Box Parameters.

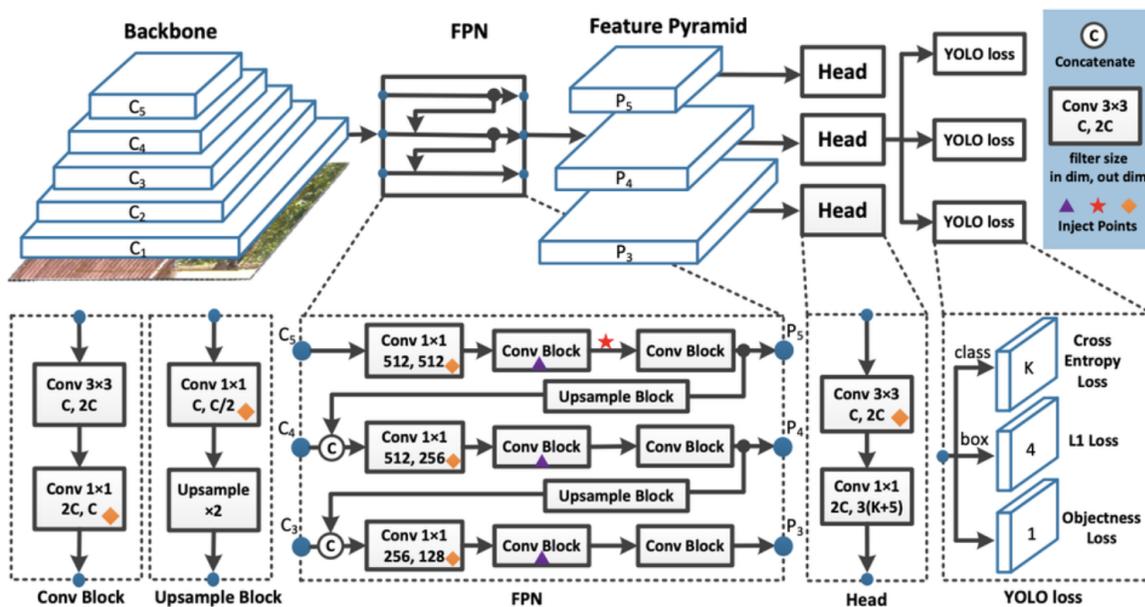


Fig. 9 Architecture of YOLO CNN Model.

object locations and classifications. This results in improved feature extraction and object detection performances.

YOLOv8 also adopts an anchor-free split Ultralytics head.

This directly predicts object locations without using predefined anchor boxes, which results in better accuracy and a more efficient detection process. YOLOv8 offers a range of

pre-trained models specific to various tasks and performance requirements, making it applicable to multiple use cases.

YOLOv8 focuses on maintaining an optimal balance between accuracy and speed. Performance-wise, YOLOv8 achieves faster inference speed than most other object detection models while maintaining a high level of accuracy.

YOLOv9

YOLOv9²¹, released in February 2024, introduces innovative approaches to overcoming the information loss challenge inherent in neural networks. The model improves learning capacity and ensures retention of crucial information through implementing programmable gradient information (PGI) and a generalized efficient layer aggregation network (GELAN).

PGI is implemented to address the information bottleneck problem. The information bottleneck principle reveals that as data passes through successive layers of a network, the potential for information loss increases. The technique uses an auxiliary reversible branch to allow for the generation of reliable gradients. This results in more accurate model updates, improving overall detection performance.

The implementation of GELAN allows YOLOv9 to improve its parameter utilization and computational efficiency. GELAN is a lightweight architecture that combines features from different network layers. This helps make YOLOv9 adaptable to a wide range of applications without sacrificing its speed or accuracy.

YOLOv10

YOLOv10²², released in May 2024 by researchers at Tsinghua University, addresses post-processing and model architecture deficiencies present in previous YOLO versions. YOLOv10 eliminates non-maximum suppression (NMS) and optimizes various model components to reduce computational overhead and improve performance.

The architecture of YOLOv10 is similar to the architecture of previous YOLO versions, but it introduces several new features. The model's architecture consists of a backbone and neck similar to those of previous models; however, it features two heads. YOLOv10 utilizes a one-to-many head that generates multiple predictions per object during training, and a one-to-one head that generates the single best prediction per object during inference.

The one-to-one head utilized during inference removes the need for NMS. YOLOv10 also incorporates a holistic efficiency-accuracy-driven model design. Efficiency enhancements include lightweight classification heads using depth-wise separable convolutions, spatial-channel decoupled down-sampling that separates spatial and channel processing to reduce information loss, and rank-guided block design that

adapts architecture based on redundancy to optimize parameter utilization. All of these techniques minimize inference latency and computational overhead while maintaining accuracy.

YOLOv11

YOLOv11²³, released in September 2024 by Ultralytics, contains improvements in architecture and training methods over previous YOLO models. YOLOv11 has many model variants, each applicable to specific tasks like pose estimation, instance segmentation, image classification, object detection, or oriented object detection.

YOLOv11 employs an enhanced backbone and neck architecture, which enhances the feature extraction capabilities of the model. YOLOv11 is optimized for efficiency and speed. YOLOv11 can also be deployed across various environments. This includes edge devices, cloud platforms, and systems supporting NVIDIA GPUs.

YOLOv11 has been able to achieve higher accuracies than some previous YOLO models with fewer parameters. When implemented on the COCO dataset, YOLOv11 achieved a higher accuracy than YOLOv8 with 22% fewer parameters.

YOLOv12

YOLOv12²⁴, released in 2025, was developed by researchers from the University of Buffalo, SUNY, and the University of Chinese Academy of Sciences. It introduces an attention-centric architecture different from the traditional CNN-based approaches used in previous YOLO models.

The attention-centric architecture YOLOv12 employs focuses on an image's important areas, rather than treating every part of the image equally. This cuts down on unnecessary processing, making the model sharper and more efficient. YOLOv12 also utilizes FlashAttention, a memory-efficient algorithm, to speed up image analysis. FlashAttention optimizes data strain and uses less memory.

YOLOv12 organizes its layers using Residual Efficient Layer Aggregation Networks (R-ELAN). R-ELAN is a feature aggregation model based on ELAN, designed to address optimization challenges. It incorporates block-level residual connections and a bottleneck-like structure. R-ELAN makes training more stable and object recognition sharper.

YOLOv12 comes in different variants optimized for different applications. Smaller versions prioritize speed, while medium and large versions strike a balance between speed and accuracy. Because of its architecture, YOLOv12 is expected to run slower than previous YOLO models like YOLOv11.

Experiments and Results

Training and Implementation Details

All models were implemented using their small variant and initialized with pretrained weights from the COCO (Common Objects in Context) dataset. The dataset was split into 72% for training, 8% for validation, and 20% for testing using random sampling with a fixed seed of 42. Training was conducted for 100 epochs with a batch size of 16 and input image dimensions of 640×640 pixels, using the NVIDIA Tesla P100 GPU. The model was trained using a Stochastic Gradient Descent optimizer configured with an initial learning rate of 0.01, momentum of 0.937, and weight decay of 0.0005. The learning rate schedule employed linear decay with a final learning rate fraction of 0.01, resulting in a final learning rate of 0.0001 by the end of training, along with a 3-epoch warmup period starting from a warmup momentum of 0.8. Data augmentation techniques included mosaic augmentation, color space adjustments (hue=0.015, saturation=0.7, value=0.4), horizontal flipping with 50% probability, random translation of $\pm 10\%$, and scaling with a gain of 0.5. Non-Maximum Suppression used an IoU threshold of 0.7 and a confidence threshold of 0.001 for validation. Each model employs a composite loss function consisting of three weighted components: bounding box regression loss with a gain of 7.5, classification loss with a gain of 0.5, and Distribution Focal Loss with a gain of 1.5 for refined box localization. The architectures use Sigmoid Linear Unit activation functions throughout the network. All experiments used the Ultralytics library, and were conducted on Kaggle, a popular platform for machine learning projects.

Two additional augmentation schemes have been implemented on YOLOv9: mixup and cutmix²⁵. Mixup blends two images and their labels to help the model identify objects when they are partially obscured or blurred. The mixup coefficient controls the probability of applying the augmentation. For YOLOv9, it has been kept as 0.1, 0.3, 0.5, 0.7, and 0.9. On the other hand, cutmix takes a rectangular region out of one training image and pastes it onto another image. This creates realistic occlusion examples for the model to train on, improving the robustness of the model. The cut mix hyperparameter controls the probability of adding the augmentation to a given training image. For YOLOv9, the cutmix coefficient has also been kept as 0.1, 0.3, 0.5, 0.7, and 0.9.

Performance Metrics

The performance metrics we have used in our research are precision, recall, Mean Average Precision (mAP), and F1 score. A confusion matrix has been made for each experiment run, and mAP has been measured at a 50% overlap threshold and a 50% to 90% overlap threshold.

Precision measures the percent of correct predictions out of all predictions made in the positive class. The formula for precision is shown in the equation below. TP denotes true positive (a real object that the detector correctly found), while FP denotes false positive (detected an object that wasn't actually there).

$$\text{Precision} = \frac{TP}{(TP + FP)} \quad (1)$$

Recall measures the percentage of objects the model detected out of the total number of objects in the dataset. The formula for recall is shown in the equation below. FN denotes false negatives (a real object that the detector missed).

$$\text{Recall} = \frac{TP}{(TP + FN)} \quad (2)$$

The Mean Average Precision (mAP) measures how well a model performs across the entire confidence spectrum. It is calculated by taking the area under the curve formed by a precision vs recall graph. Averaging out the area under this curve for all classes will provide the mAP.

The last metric used is F1 score. F1 score calculates the harmonic mean of precision and recall. It provides a balanced score between the precision and recall. The formula for F1 score is shown in the equation below.

$$F1 = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})} \quad (3)$$

All of these relationships can be viewed using a confusion matrix.

Performance of different YOLO models

Our experiments showed that YOLOv7 achieved an overall precision of 82.1%, a recall of 74.3%, a mAP50 of 80.4%, and a mAP50-95 of 57.4%. It took 3.062 hours to train for 100 epochs, and resulted in an F1 score of 0.780. Table 1 further shows the class-wise performance of YOLOv7. The model's detection pipeline often fails between the detection and classification stages. 837 of the real objects the model detected were classified as background during the final prediction, but the model wrongly predicted 387 objects. This suggests the model's NMS is simultaneously under-suppressing false positives and over-suppressing valid detections. This is likely because of bounding box regression errors and an inconsistent application of IoU (Intersection over Union) thresholds, which is the measure of how much predicted boxes overlap with each other to decide which duplicates to remove. The bounding boxes that YOLOv7 draws are slightly misaligned, so when the NMS tries to remove duplicates, it accidentally deletes the real object while keeping the fake one.

Table 1 Class-wise Performance of YOLOv7

Class	Images	Instances	Precision	Recall	mAP
All	749	5099	0.821	0.743	0.804
Car	662	2794	0.911	0.915	0.957
Pedestrian	183	445	0.856	0.674	0.784
Van	232	307	0.901	0.893	0.937
Cyclist	114	167	0.833	0.784	0.852
Truck	115	118	0.953	0.932	0.980
Misc	80	105	0.917	0.771	0.835
Tram	36	54	0.829	0.907	0.935
Person_sitting	10	23	0.587	0.652	0.677
DontCare	524	1085	0.591	0.166	0.287

Table 2 Class-wise Performance of YOLOv8

Class	Images	Instances	Precision	Recall	mAP
All	749	5300	0.861	0.746	0.819
Car	659	2906	0.942	0.903	0.964
Pedestrian	199	488	0.908	0.629	0.784
Van	213	311	0.934	0.923	0.961
Cyclist	124	177	0.863	0.780	0.816
Truck	96	101	0.979	0.980	0.992
Misc	72	96	0.880	0.812	0.907
Tram	43	71	0.850	0.877	0.916
Person_sitting	15	46	0.729	0.652	0.715
DontCare	538	1104	0.659	0.137	0.313

Table 3 Class-wise Performance of YOLOv9

Class	Images	Instances	Precision	Recall	mAP
All	749	5402	0.848	0.848	0.868
Car	681	3008	0.924	0.968	0.982
Pedestrian	161	424	0.882	0.818	0.860
Van	233	319	0.934	0.978	0.986
Cyclist	100	134	0.871	0.908	0.945
Truck	122	129	0.976	1.000	0.995
Misc	70	89	0.861	0.933	0.951
Tram	28	34	0.893	1.000	0.993
Person_sitting	10	15	0.619	0.759	0.734
DontCare	562	1250	0.509	0.474	0.428

YOLOv8 achieved an overall precision of 86.1%, a recall of 74.6%, a mAP50 of 81.9%, and a mAP50-95 of 59.2%. The YOLOv8 model took 2.805 hours to train on its 100 epochs, and resulted in an F1 score of 0.799. Table 2 shows the class-wise performance of YOLOv8. The model exhibits severe object deletion syndrome, erasing 899 real objects while creating only 395 phantom objects, resulting in a net destruction of 504 visual objects from scenes. The widespread object disappearance suggests YOLOv8's anchor-free detection head success-

fully identifies objects, but the model loses confidence during final classification, causing detected objects to be reclassified as background due to overly conservative confidence thresholds. The technical issues stem from YOLOv8's NMS implementation over-suppressing valid detections while simultaneously allowing false positives to persist, indicating poor calibration of IoU thresholds and the split Ultralytics head's confidence scoring mechanism.

YOLOv9 achieved an overall precision of 84.8%, a recall of

Table 4 Class-wise Performance of YOLOv10

Class	Images	Instances	Precision	Recall	mAP
All	749	5224	0.801	0.746	0.803
Car	683	2908	0.871	0.931	0.962
Pedestrian	176	482	0.744	0.712	0.763
Van	199	266	0.867	0.907	0.952
Cyclist	111	164	0.773	0.762	0.834
Truck	101	106	0.955	0.981	0.992
Misc	69	88	0.879	0.807	0.876
Tram	30	43	0.866	0.904	0.951
Person_sitting	12	26	0.719	0.493	0.616
DontCare	513	1141	0.528	0.220	0.279

Table 5 Class-wise Performance of YOLOv11

Class	Images	Instances	Precision	Recall	mAP
All	749	5237	0.839	0.768	0.818
Car	679	2888	0.905	0.928	0.965
Pedestrian	184	422	0.814	0.713	0.786
Van	227	306	0.900	0.918	0.963
Cyclist	123	171	0.861	0.825	0.901
Truck	104	111	0.973	0.910	0.938
Misc	93	121	0.884	0.822	0.878
Tram	39	58	0.924	0.948	0.972
Person_sitting	6	15	0.662	0.654	0.649
DontCare	540	1145	0.620	0.194	0.305

Table 6 Class-wise Performance of YOLOv12

Class	Images	Instances	Precision	Recall	mAP
All	749	5193	0.852	0.759	0.826
Car	662	2881	0.935	0.928	0.971
Pedestrian	197	490	0.884	0.734	0.834
Van	215	277	0.951	0.903	0.953
Cyclist	117	168	0.839	0.792	0.846
Truck	96	101	0.955	0.950	0.988
Misc	78	94	0.874	0.809	0.884
Tram	37	47	0.874	0.884	0.915
Person_sitting	16	31	0.719	0.613	0.720
DontCare	521	1104	0.635	0.207	0.324

Table 7 Overall Performance of YOLO Models

Model	Precision	Recall	mAP	Latency(ms)	F1 Score
YOLOv7	0.821	0.743	0.804	2.4	0.780
YOLOv8	0.861	0.746	0.819	2.6	0.799
YOLOv9	0.848	0.848	0.868	8.6	0.848
YOLOv10	0.801	0.746	0.803	1.5	0.773
YOLOv11	0.839	0.768	0.818	2.5	0.802
YOLOv12	0.852	0.759	0.826	2.6	0.803

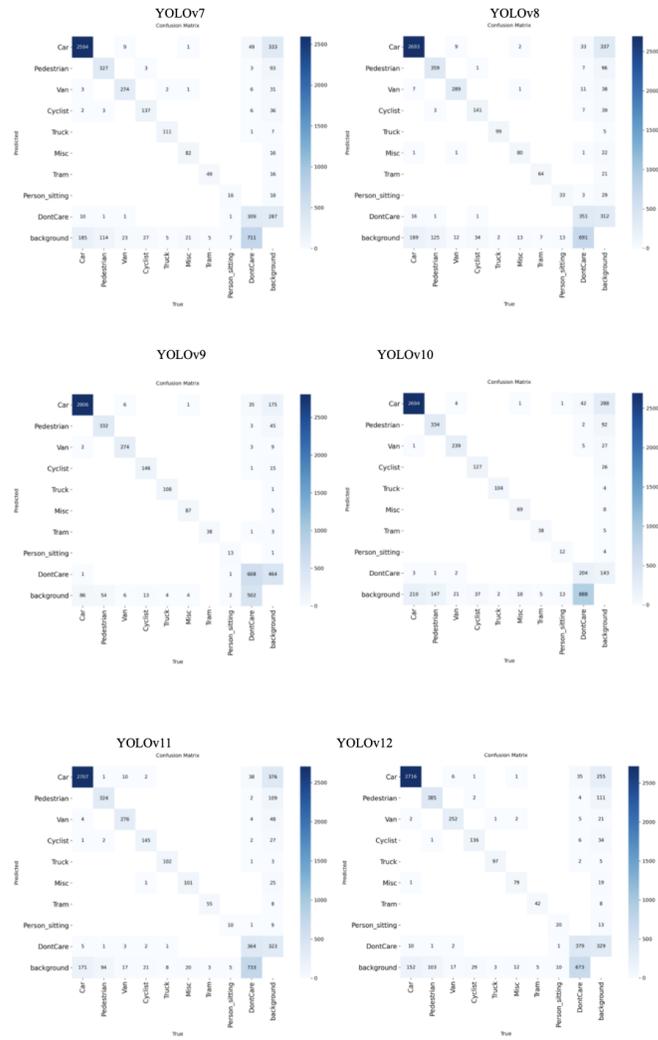


Fig. 10 Confusion Matrices of YOLO Models.



Fig. 11 Sample Prediction Images from YOLOv9.

84.8%, a mAP50 of 86.8%, and a mAP50-95 of 69.6%. The YOLOv9 model took 9.468 hours to train on its 100 epochs, and resulted in an F1 score of 0.848. Table 3 shows the class-wise performance of YOLOv9. The model's PGI mechanism successfully addresses information bottleneck problems for most object categories, with 6 classes showing less than 10% erasure rates, indicating strong information preservation during forward propagation. YOLOv9's GELAN architecture proves highly efficient with 4 classes, achieving less than 90% identity stability. Despite these improvements, the model maintains conservative classification behavior with certain challenging classes, like Person_sitting, while creating only 197 phantom objects. This suggests YOLOv9 balances precision and recall effectively in its detection pipeline.

YOLOv10 achieved an overall precision of 80.1%, a recall of 74.6%, a mAP50 of 80.3% and a mAP50-95 of 58.5%. The YOLOv10 model took 3.219 hours to train on its 100 epochs, and resulted in an F1 score of 0.773. Table 4 shows the class-wise performance of YOLOv10. The model achieved 3,821 correct object detections, but predicted 453 phantom objects and did not detect 597 objects. The efficiency benefits offered by YOLOv10's elimination of NMS come at the cost of worsened precision and recall. This model performed the worst out of all models tested, suggesting NMS is needed for object detection consistency and accuracy.

YOLOv11 achieved an overall precision of 83.9%, a recall of 76.8%, a mAP50 of 81.8% and a mAP50-95 of 60.9%. The YOLOv11 model took 3.227 hours to train on its 100 epochs, and resulted in an F1 score of 0.802. Table 5 shows the class-wise performance of YOLOv11. The model correctly predicts 4,817 objects, but decreases, erases 928, and predicts 339 phantom objects. This model struggles most with background detection and differentiating between different types of vehicles. 38 cars are classified as background objects, and 10 cars are classified as vans. This suggests YOLOv11 prioritizes computational efficiency over feature discrimination, resulting in the model being unable to distinguish between similar object classes.

YOLOv12 achieved an overall precision of 85.2%, a recall of 75.9%, a mAP50 of 82.6% and a mAP50-95 of 60.6%. The YOLOv12 model took 3.269 hours to train on its 100 epochs, and resulted in an F1 score of 0.803. Table 6 shows the class-wise performance of YOLOv12. The model correctly predicts 4,779 objects, while erasing 795 and creating 331. The model's performance is very class-dependent, with only 3 classes achieving less than 15% erasure rates. The model simultaneously over-attends to background regions while under-attending to real objects, similarly to most of the other YOLO models tested. 152 phantom cars are predicted, 102 phantom pedestrians are predicted, 35 cars are classified as background, 10 DontCare instances are classified as cars, and 6 cars are classified as pedestrians. This suggests that

YOLOv12's attention-centric architecture fails to solve the object detection problems encountered in previous models, like YOLOv11.

Table 7 displays the overall performance of each YOLO model together. YOLOv8 achieved the highest precision, but did not perform exceptionally well in recall. YOLOv9

achieved the highest recall while maintaining a high precision, so it attained the highest F1 score. The models that performed the next best were YOLOv12 and YOLOv11, both achieving F1 scores 0.045 and 0.044 lower than YOLOv9, respectively. The median latency is 2.55 milliseconds per image. Despite its exceptional performance, YOLOv9's latency is almost three times as much as this median. The models that provide a more balanced ratio of latency and performance are YOLOv11 and YOLOv12. All six models show similar class-wise trends. Each model performs best on larger objects, like trucks, cars, and vans. Smaller objects like people and cyclists have the worst performances. Every model's best performing class is (Table 7) Truck, while their worst performing class is Person_sitting. Fig. 10 displays the confusion matrices for the 6 models, and Fig. 11 depicts sample prediction images from YOLOv9.

Impact of Data Augmentation Techniques

Effect of Mixup Augmentation

When YOLOv9 ran with a mixup augmentation coefficient of 0.1, it achieved an overall precision of 89.3%, a recall of 86.0%, a mAP50 of 89.6%, a mAP50-95 of 71.5%, and a F1 score of 0.876. This implementation of mixup showed a 4.5% increase in precision, a 1.2% increase in recall, and an improvement in F1 score of .28. Table 8 shows the class-wise performance of implementing mixup at this value. The non-vehicle classes saw major improvements in precision: Person_sitting increased by 23.1%, Pedestrian increased by 3.2%, and Cyclist increased by 4.8%. The recall for the Truck and Van classes dropped by 3.6% and 2.0% respectively, yet both classes' recalls remained over 95%.

When the model ran with a mixup coefficient of 0.3, it achieved an overall precision of 89.8%, a recall of 83.0%, a mAP50 of 88.7%, a mAP50-95 of 71.8%, and a F1 score of

0.863. Table 9 shows the class-wise performance of implementing mixup at this value. While the overall performance dropped when increasing the mixup to 0.3, the Pedestrian class saw improvements. The precision of this class improved by 1.5% and the model achieved 404 correct detections. However, the Cyclist class dropped in precision by 4.0% and background confusion increased. Across several classes, there were more false positives than before, suggesting the higher mixup coefficient may be making the model too conservative in certain cases.

Table 8 Class-wise Performance of Mixup Coefficient 0.1

Class	Images	Instances	Precision	Recall	mAP
All	749	5137	0.893	0.860	0.896
Car	675	2895	0.942	0.966	0.981
Pedestrian	174	386	0.914	0.845	0.909
Van	213	286	0.959	0.958	0.978
Cyclist	111	159	0.919	0.906	0.945
Truck	109	112	0.988	0.964	0.975
Misc	74	92	0.960	0.957	0.976
Tram	27	38	0.901	1.000	0.991
Person_sitting	7	16	0.850	0.750	0.881
DontCare	536	1153	0.598	0.395	0.429

Table 9 Class-wise Performance of Mixup Coefficient 0.3

Class	Images	Instances	Precision	Recall	mAP
All	749	5030	0.898	0.830	0.887
Car	655	2783	0.941	0.967	0.982
Pedestrian	188	481	0.929	0.819	0.889
Van	208	293	0.949	0.951	0.978
Cyclist	103	144	0.880	0.840	0.905
Truck	108	117	0.989	0.957	0.987
Misc	67	85	0.903	0.873	0.916
Tram	29	40	0.961	0.975	0.994
Person_sitting	6	13	0.706	0.846	0.904
DontCare	516	1074	0.610	0.365	0.446

Table 10 Class-wise Performance of Mixup Coefficient 0.5

Class	Images	Instances	Precision	Recall	mAP
All	749	5185	0.887	0.868	0.892
Car	664	2943	0.942	0.969	0.985
Pedestrian	179	391	0.896	0.849	0.912
Van	213	287	0.958	0.969	0.988
Cyclist	107	153	0.929	0.850	0.908
Truck	110	119	0.957	0.983	0.993
Misc	81	99	0.892	0.919	0.930
Tram	27	37	0.935	1.000	0.968
Person_sitting	10	32	0.851	0.875	0.893
DontCare	526	1124	0.624	0.396	0.452

When the model ran with a mixup coefficient of 0.5, it achieved an overall precision of 88.7%, a recall of 86.8%, a mAP50 of 89.2%, a mAP50-95 of 71.3%, and a F1 score of 0.877. Table 10 shows the class-wise performance of implementing mixup at this value. Overall performance increased compared to the last two experiments. The classes that benefited from the new mixup coefficient the most were the vehicle classes: Car, Van, and Truck. Person_sitting also saw major improvements, as it achieved 29 correct predictions, 20 more

than the model did without augmentation. The Cyclist class, however, had a drop in recall with minimal improvement in precision, and background false positives also decreased.

When the model ran with a mixup coefficient of 0.7, it achieved an overall precision of 86.0%, a recall of 86.3%, a mAP50 of 88.7%, a mAP50-95 of 71.3%, and a F1 score of 0.861. Table 11 shows the class-wise performance of implementing mixup at this value. When increasing the augmentation to 0.7, there was an overall decline; however, some classes

Table 11 Class-wise Performance of Mixup Coefficient 0.7

Class	Images	Instances	Precision	Recall	mAP
All	749	5172	0.860	0.863	0.887
Car	659	2883	0.949	0.959	0.982
Pedestrian	182	403	0.895	0.804	0.885
Van	208	296	0.931	0.965	0.972
Cyclist	120	170	0.906	0.910	0.950
Truck	107	114	0.979	0.982	0.994
Misc	69	82	0.863	0.963	0.971
Tram	34	43	0.872	0.952	0.916
Person_sitting	8	21	0.790	0.810	0.908
DontCare	552	1160	0.557	0.422	0.407

Table 12 Class-wise Performance of Mixup Coefficient 0.9

Class	Images	Instances	Precision	Recall	mAP
All	749	5075	0.889	0.867	0.892
Car	661	2793	0.948	0.966	0.984
Pedestrian	190	456	0.893	0.842	0.902
Van	220	301	0.964	0.963	0.982
Cyclist	114	161	0.941	0.890	0.949
Truck	83	86	0.997	1.000	0.995
Misc	79	94	0.901	0.973	0.972
Tram	37	52	0.928	0.962	0.989
Person_sitting	8	10	0.799	0.795	0.784
DontCare	529	1122	0.633	0.411	0.467

Table 13 Overall Performance of Mixup Values

Model	Precision	Recall	mAP	Latency(ms)	F1 Score
No Augmentation	0.848	0.848	0.868	8.6	0.848
Mixup 0.1	0.893	0.860	0.896	10.2	0.876
Mixup 0.3	0.898	0.830	0.887	6.6	0.863
Mixup 0.5	0.887	0.868	0.892	8.9	0.877
Mixup 0.7	0.860	0.863	0.887	10.6	0.861
Mixup 0.9	0.889	0.867	0.892	6.8	0.878

Table 14 Class-wise Performance of Cutmix Coefficient 0.1

Class	Images	Instances	Precision	Recall	mAP
All	749	4969	0.901	0.855	0.887
Car	671	2712	0.947	0.963	0.983
Pedestrian	170	425	0.880	0.821	0.891
Van	193	260	0.954	0.965	0.982
Cyclist	122	158	0.930	0.836	0.908
Truck	99	104	0.976	0.981	0.995
Misc	77	95	0.943	0.958	0.972
Tram	37	42	0.970	1.000	0.995
Person_sitting	15	29	0.919	0.787	0.856
DontCare	544	1144	0.589	0.381	0.403

still saw improvements. The classes that peaked with this implementation were Cyclist, with 157 correct predictions, and Tram, with 42 correct predictions. Every other class saw regression, with Pedestrian having its worst performance in this experiment. This implementation also resulted in increased background confusion, as the model is getting less confident about what instances are objects or backgrounds.

When the model ran with a mixup coefficient of 0.9, it achieved an overall precision of 88.9%, a recall of 86.7%, a mAP50 of 89.2%, a mAP50-95 of 72.4%, and a F1 score of 0.878. Table 12 shows the class-wise performance of implementing mixup at this value. The classes that peaked in this implementation were Truck, Car, Van, and Cyclist. While they did not peak in this experiment, other classes still had far better performance than they did when no augmentation was applied to the model. This implementation also resulted in reduced background confusion, suggesting the model is getting more confident in identifying objects.

Table 13 displays the overall performance of each mixup coefficient together. When the coefficient was 0.1, the model achieved the highest precision, and when it was implemented at 0.5, it achieved its highest recall. When the coefficient was 0.9, it struck a better balance between precision and recall, resulting in the highest F1 score.

Compared to when the model had no augmentation, implementing mixup at 0.9 resulted in a 4.1% increase in precision, a 1.9% increase in recall, and a F1 score increase of 0.030, a considerable improvement. All classes benefited from the augmentation, but the non-vehicle classes benefited the most. The Person_sitting, Pedestrian, and Cycling classes all saw substantial improvements when subjected to severe augmentation, while the Car, Truck, and Van classes saw minimal improvements. Since the non-vehicle classes are less frequent in the dataset and contain smaller objects, this suggests that mixup augmentation helps improve precision and recall on instances that show up less frequently or are smaller. Fig. 12 displays the confusion matrices for all implementations of mixup.

Effect of Cutmix Augmentation

When YOLOv9 ran with a cutmix coefficient of 0.1, it achieved an overall precision of 90.1%, a recall of 85.5%, a mAP50 of 88.7%, a mAP50-95 of 70.9%, and a F1 score of 0.877. Table 14 shows the class-wise performance of implementing cutmix at this value. The non-vehicle classes saw major improvements in precision: Person_sitting increased by 30.0%, Pedestrian remained stable with a slight decline of 0.2%, and Cyclist increased by 5.9%. Person_sitting achieved 19 correct detections, more than doubling from 9 with no augmentation. Background confusion decreased significantly across most classes, with 122 fewer total background misclassifications compared to no augmentation.

When the model ran with a cutmix coefficient of 0.3, it achieved an overall precision of 88.8%, a recall of 85.2%, a mAP50 of 89.0%, a mAP50-95 of 71.6%, and a F1 score of 0.870. Table 15 shows the class-wise performance of implementing cutmix at this value. This augmentation level achieved the highest mAP50 score across all cutmix coefficients. The Pedestrian class saw significant improvements, achieving 369 correct detections compared to 344 with no augmentation, representing a 7.3% increase. Person_sitting maintained its improved performance with 19 correct detections. However, some vehicle classes, like Van, showed slight performance decreases compared to lighter augmentation levels.

When the model ran with a cutmix coefficient of 0.5, it achieved an overall precision of 88.2%, a recall of 85.1%, a mAP50 of 87.9%, a mAP50-95 of 71.3%, and a F1 score of 0.866. Table 16 shows the class-wise performance of implementing cutmix at this value. Performance began to decline compared to lighter augmentation levels, though it still maintained improvements over no augmentation. Person_sitting dropped to 14 correct detections, showing sensitivity to higher cutmix coefficients. Vehicle classes remained relatively stable, with Car maintaining 98.2% mAP50. Background confusion increased slightly compared to 0.1 and 0.3, with 67 fewer background misclassifications than with no augmentation.

When the model ran with a cutmix coefficient of 0.7, it achieved an overall precision of 85.2%, a recall of 82.7%, a mAP50 of 85.1%, a mAP50-95 of 68.8%, and a F1 score of 0.839. Table 17 shows the class-wise performance of implementing cutmix at this value. This represented the worst overall performance across all cutmix values, with significant declines in most metrics. Person_sitting performance collapsed to just 4 correct detections with 53.4% mAP50, suggesting over-augmentation. However, some classes, like Truck, achieved a perfect 100% recall, and Tram maintained strong performance with 99.5% mAP50. Background confusion patterns remained similar to 0.5, but the overall detection confidence decreased substantially.

When the model ran with a cutmix coefficient of 0.9, it achieved an overall precision of 86.6%, a recall of 86.9%, a mAP50 of 89.1%, a mAP50-95 of 71.2%, and a F1 score of 0.867. Table 18 shows the class-wise performance of implementing cutmix at this value. Performance recovered significantly from the 0.7 dip, showing the highest recall across all cutmix values. Person_sitting achieved remarkable recovery with 19 correct detections and 94.0% mAP50, its best performance across all augmentation levels. The model achieved the best balance between precision and recall at this extreme augmentation level. Background confusion decreased by 94 fewer misclassifications than with no augmentation, suggesting improved object-background distinction.

Table 19 displays the overall performance of each cutmix value together. When cutmix was implemented at 0.1, the

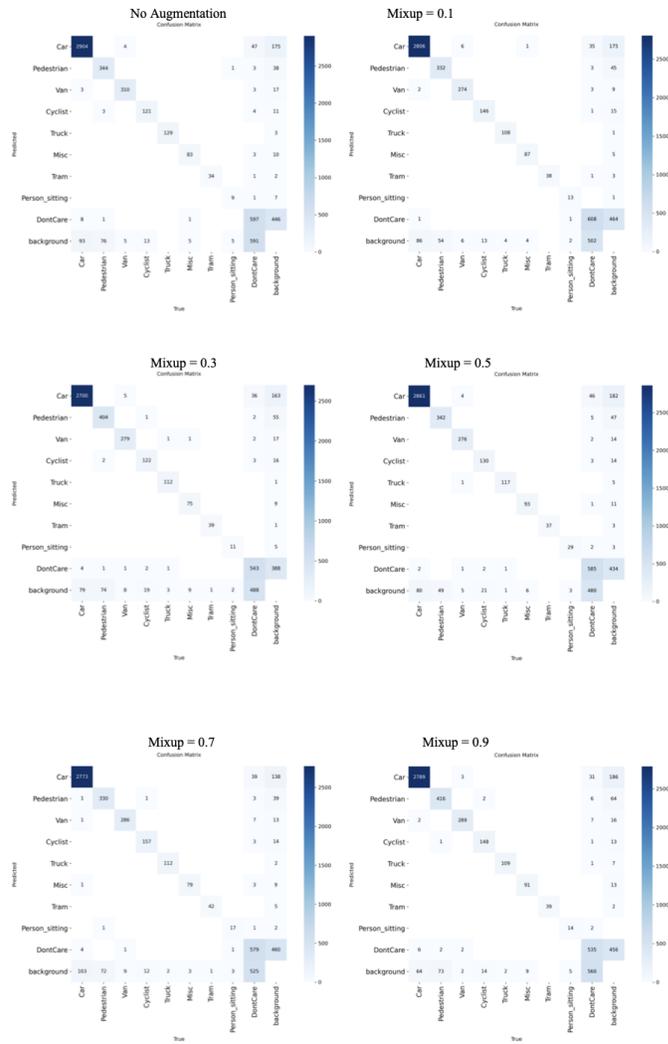


Fig. 12 Confusion Matrices for Mixup Value.

Table 15 Class-wise Performance of Cutmix Coefficient 0.3

Class	Images	Instances	Precision	Recall	mAP
All	749	5224	0.888	0.852	0.890
Car	676	2941	0.941	0.963	0.981
Pedestrian	169	427	0.888	0.796	0.868
Van	221	306	0.928	0.967	0.979
Cyclist	118	172	0.943	0.895	0.939
Truck	95	103	0.963	0.961	0.986
Misc	92	113	0.915	0.903	0.948
Tram	30	51	0.983	0.980	0.994
Person_sitting	10	21	0.826	0.810	0.880
DontCare	529	1090	0.603	0.396	0.435

Table 16 Class-wise Performance of Cutmix Coefficient 0.5

Class	Images	Instances	Precision	Recall	mAP
All	749	5191	0.882	0.851	0.879
Car	664	2882	0.939	0.965	0.982
Pedestrian	179	434	0.888	0.837	0.894
Van	225	303	0.953	0.957	0.969
Cyclist	111	171	0.912	0.871	0.925
Truck	108	115	0.960	0.983	0.991
Misc	69	79	0.920	0.937	0.964
Tram	32	56	0.988	0.982	0.994
Person_sitting	11	20	0.788	0.700	0.767
DontCare	525	1131	0.590	0.429	0.426

Table 17 Class-wise Performance of Cutmix Coefficient 0.7

Class	Images	Instances	Precision	Recall	mAP
All	749	5157	0.852	0.827	0.851
Car	664	2864	0.951	0.969	0.982
Pedestrian	169	410	0.903	0.827	0.886
Van	211	277	0.920	0.989	0.986
Cyclist	116	169	0.974	0.870	0.926
Truck	94	97	0.974	1.000	0.988
Misc	68	90	0.881	0.905	0.956
Tram	36	49	0.999	1.000	0.995
Person_sitting	4	8	0.497	0.500	0.534
DontCare	547	1193	0.573	0.386	0.410

Table 18 Class-wise Performance of Cutmix Coefficient 0.9

Class	Images	Instances	Precision	Recall	mAP
All	749	5239	0.866	0.869	0.891
Car	673	2950	0.932	0.971	0.983
Pedestrian	184	442	0.898	0.812	0.879
Van	219	277	0.937	0.971	0.978
Cyclist	110	165	0.910	0.891	0.926
Truck	95	104	0.958	1.000	0.993
Misc	86	97	0.916	0.904	0.938
Tram	29	46	0.901	1.000	0.980
Person_sitting	12	20	0.775	0.900	0.940
DontCare	529	1138	0.565	0.368	0.397

Table 19 Overall Performance of Cutmix Values

Model	Precision	Recall	mAP	Latency(ms)	F1 Score
No Augmentation	0.848	0.848	0.868	8.6	0.848
Mixup 0.1	0.901	0.855	0.887	6.7	0.877
Mixup 0.3	0.888	0.852	0.893	8.9	0.870
Mixup 0.5	0.897	0.848	0.881	8.5	0.872
Mixup 0.7	0.861	0.824	0.846	10.1	0.842
Mixup 0.9	0.891	0.847	0.891	8.6	0.868

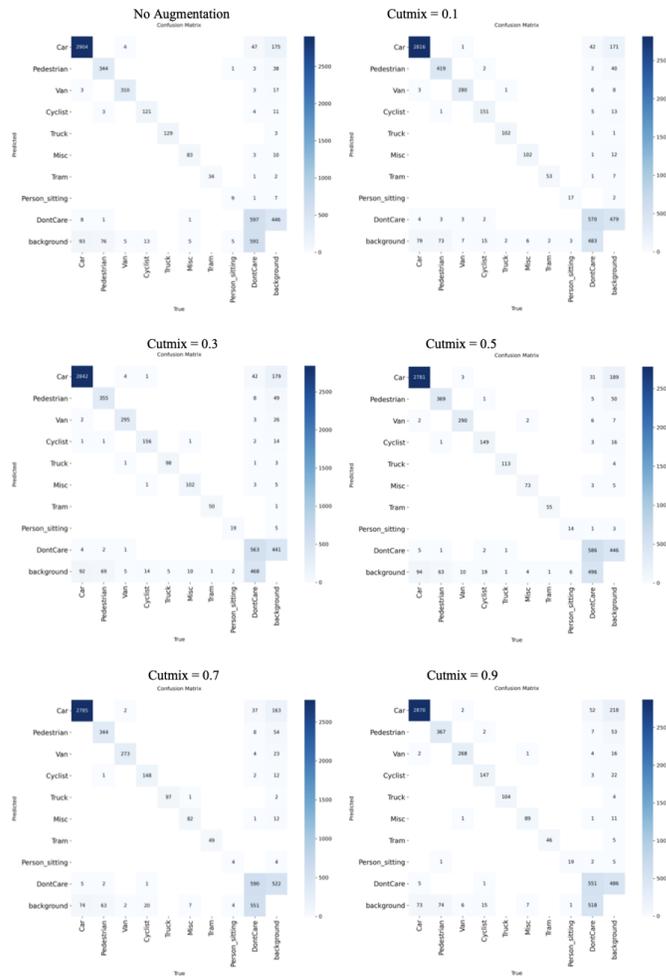


Fig. 13 Confusion Matrices for Cutmix Coefficients.

model achieved the highest precision at 90.1%, and when implemented at 0.9, it achieved the highest recall at 86.9%. The model achieved the highest mAP50 of 89.0% at cutmix 0.3, while cutmix 0.1 resulted in the highest F1 score of 0.877. Compared to when the model had no augmentation, implementing cutmix at 0.1 resulted in a 7.1% increase in precision, though with a 1.6% decrease in recall, and a F1 score increase of 0.027. All classes benefited from cutmix augmentation, but the non-vehicle classes benefited the most dramatically. The Person_sitting class showed the most substantial improvements, achieving over double the correct detections with cutmix compared to no augmentation. Since the non-vehicle classes are less frequent in the dataset and contain smaller objects, this suggests that cutmix augmentation particularly helps improve performance on instances that show up less frequently or are smaller, likely due to the occlusion-based training that cutmix provides. Fig. 13 displays the con-

fusion matrices for all implementations of cutmix.

Conclusions and Future Work

In this paper, we set out to improve object detection models for car safety features to reduce the number of accidents and fatalities that occur on roads. To do this, we compared 6 different YOLO models and investigated how we can improve them with data augmentation schemes. It was found that the model that achieved the highest overall performance was YOLOv9, which achieved a precision of 84.8%, a recall of 84.8%, and an F1 score of 84.8%. Implementing a mixup factor of 0.9 and a cutmix factor of 0.1 improved the performance of the model the most. The mixup factor of 0.9 improves the model's precision by 4.1% and its recall by 1.9%, resulting in a F1 score increase of 3.0%. The cutmix factor of 0.1 increases the model's precision by 5.3% and its recall by 0.7%, resulting in

a F1 score increase of 2.9%.

Our objective of determining the most accurate object detection and classification model, and finding the best ways to improve it, was achieved: To attain the highest accuracy and precision when detecting roadside objects, YOLOv9 must be used with a mixup factor of 0.9 or a cutmix factor of 0.1. The implementation of this more precise and accurate model could transform automotive safety standards. It provides drivers with enhanced awareness of their surroundings, instead of relying on their reaction times and potentially impaired judgment. This will heavily reduce car accidents and injuries, allowing drivers to feel safer in their cars.

Several areas can be investigated further. First, the work can be extended to include other object detection models alongside YOLO. Second, more training parameters other than augmentation schemes can be used to improve the model. Third, real-time object detection devices can be built into vehicles using improved models to help prevent accidents and fatalities. There were also several limitations in this study. First, the study's scope was constrained to controlled dataset conditions without accounting for environmental variables like fog, rain, or nighttime scenarios that commonly affect real-world driving. Additionally, performance evaluation relied on standard metrics without considering edge cases or failure modes that could prove critical in actual deployment scenarios. The next critical step is transitioning these laboratory findings into production vehicles, where this optimized object detection system could significantly reduce the 1.2 million annual traffic fatalities caused by delayed driver reactions.

References

- 1 World Health Organization, *Road traffic injuries*, World Health Organization (WHO), Fact sheet, 2023, <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>, Retrieved July 16, 2025.
- 2 U.S. Department of Transportation and National Highway Traffic Safety Administration, *2016 Fatal Motor Vehicle Crashes: Overview*, National Highway Traffic Safety Administration Technical Report DOT HS 812 456, 2017.
- 3 V. Winland, *What Is Feature Extraction?*, IBM, 2025, <https://www.ibm.com/think/topics/feature-extraction>.
- 4 K. Li, W. Ma, U. Sajid, Y. Wu and G. Wang, *Object Detection with Convolutional Neural Networks*, arXiv, 2019, <https://arxiv.org/abs/1912.01844>.
- 5 A. Geiger, P. Lenz, C. Stiller and R. Urtasun, *The International Journal of Robotics Research*, 2013, **32**, 1231–1237.
- 6 S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao and F. Shen, *Image Data Augmentation for Deep Learning: A Survey*, arXiv, 2022, <https://arxiv.org/abs/2204.08610>.
- 7 W. Chen, J. Yan, W. Huang *et al.*, *Security and Safety*, 2024, **3**, 2024002.
- 8 L. K. Sahoo and V. Varadarajan, *Complex Engineering Systems*, 2025, **5**, 2.
- 9 Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu and J. Chen, *DETRs Beat YOLOs on Real-time Object Detection*, arXiv, 2023, <https://arxiv.org/abs/2304.08069>.
- 10 T. Vu, B. Sun, B. Yuan, A. Ngai, Y. Li and J. Frahm, *Supervision Interpolation via LossMix: Generalizing Mixup for Object Detection and Beyond*, arXiv, 2023, <https://arxiv.org/abs/2303.10343>.
- 11 H. Zhao, D. Sheng, J. Bao, D. Chen, D. Chen, F. Wen, L. Yuan, C. Liu, W. Zhou, Q. Chu, W. Zhang and N. Yu, *Proceedings of the 40th International Conference on Machine Learning*, 2023, pp. 42098–42109.
- 12 X. Jia, Y. Tong, H. Qiao *et al.*, *Scientific Reports*, 2023, **13**, 9711.
- 13 Y. Zhou, S. Wen, D. Wang, J. Meng, J. Mu and R. Irampaye, *Sensors*, 2022, **22**, 3349.
- 14 J. Wei, A. As'arry, K. Anas Md Rezali, M. Zuhri Mohamed Yusoff, H. Ma and K. Zhang, *IEEE Access*, 2025, **13**, 93688–93711.
- 15 J. Redmon, S. Divvala, R. Girshick and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, arXiv, 2015, <https://arxiv.org/abs/1506.02640>.
- 16 G. Jocher and Z. Keita, *YOLO Object Detection Explained*, DataCamp, 2024, <https://www.datacamp.com/blog/yolo-object-detection-explained>.
- 17 R. Kindu, *YOLO: Algorithm for Object Detection Explained [+Examples]*, V7 Labs, 2023, <https://www.v7labs.com/blog/yolo-object-detection>.
- 18 C. Wang, A. Bochkovskiy and H. Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, arXiv, 2022, <https://arxiv.org/abs/2207.02696>.
- 19 G. Boesch, *YOLOv7: A Powerful Object Detection Algorithm*, viso.ai, 2023, <https://viso.ai/deep-learning/yolov7-guide/>.
- 20 D. Reis, J. Kupec, J. Hong and A. Daoudi, *Real-Time Flying Object Detection with YOLOv8*, arXiv, 2023, <https://arxiv.org/abs/2305.09972>.
- 21 C. Wang, I. Yeh and H. Liao, *YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information*, arXiv, 2024, <https://arxiv.org/abs/2402.13616>.
- 22 A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han and G. Ding, *YOLOv10: Real-Time End-to-End Object Detection*, arXiv, 2024, <https://arxiv.org/abs/2405.14458>.
- 23 R. Khanam and M. Hussain, *YOLOv11: An Overview of the Key Architectural Enhancements*, arXiv, 2024, <https://arxiv.org/abs/2410.17725>.
- 24 Y. Tian, Q. Ye and D. Doermann, *YOLOv12: Attention-Centric Real-Time Object Detectors*, arXiv, 2025, <https://arxiv.org/abs/2502.12524>.
- 25 G. Jocher, *Data Augmentation using Ultralytics YOLO*, Ultralytics YOLO Docs, 2025, <https://docs.ultralytics.com/guides/yolo-data-augmentation/>.