A Comprehensive Evaluation of YOLO11n Optimization Techniques for Driver Drowsiness Detection

Achmad Rohmandzoni[1], Aulia Riza Mufita[1], Nada Nadhira Najwa Mazaya[1], Fitra Abdurrachman Bachtiar[1⋆], and Novanto Yudistira[1]

Informatics Engineering Department, Faculty of Computer Science, Universitas Brawijaya, Malang, Indonesia
fitra.bachtiar@ub.ac.id

Abstract. Driver drowsiness is a major contributor to road traffic accidents, highlighting the need for accurate and computationally efficient detection systems suitable for real-time deployment on resource-constrained edge devices. This study presents a comprehensive evaluation of YOLO11n optimization strategies for driver drowsiness detection using the NTHU Driver Drowsiness Detection (NTHU-DDD) dataset. The investigated approaches include TensorRT-based quantization (FP16 and INT8) and unstructured magnitude pruning (10%, 20%, and 30% sparsity). All models were trained under identical hyperparameter settings and evaluated using detection accuracy and computational efficiency metrics, including precision, recall, mean Average Precision (mAP), inference speed, and model size. Experimental results demonstrate that each optimization strategy offers distinct trade-offs between accuracy and efficiency. FP16 quantization provided the best balance between detection performance and real-time inference, achieving the highest throughput of 35.91 FPS while maintaining an mAP@50 of 0.9721. INT8 quantization produced the smallest model size (4.80 MB) with only a marginal reduction in detection accuracy, making it suitable for memory-constrained edge devices. Moderate pruning levels (10% and 20%) slightly improved generalization, whereas 30% sparsity caused substantial performance degradation. Overall, FP16 TensorRT quantization emerged as the most practical optimization strategy for real-time edge deployment, offering an effective balance between detection accuracy and computational efficiency. These findings provide practical guidance for developing efficient driver monitoring systems for intelligent transportation applications.

Keywords: Driver drowsiness detection · YOLO11n · Pruning · Quantization · Ghost Module · Edge computing.

⋆ Corresponding author.

© The Author(s) 2026
M. Murtadha et al. (eds.), *Proceedings of the 2026 11th International Conference on Sustainable Information Engineering and Technology (SIET 2026)*, Atlantis Highlights in Computer Sciences 26,
https://doi.org/10.2991/978-94-6239-828-3_8

1 Introduction

Driver drowsiness is a major contributor to road traffic accidents worldwide. According to the World Health Organization (WHO), approximately 1.19 million people die annually in road traffic accidents, with driver drowsiness being one of the leading contributing factors [1]. This highlights the urgent need for reliable and efficient driver drowsiness detection systems, particularly for Advanced Driver Assistance Systems (ADAS) and autonomous vehicle applications.

Computer vision and deep learning have demonstrated significant potential for driver drowsiness detection through facial image analysis. Compared with physiological sensor-based methods, vision-based approaches require only a camera, making them more practical for real-world deployment. Among object detection models, the YOLO (You Only Look Once) family is widely adopted because it provides a favorable balance between detection accuracy and inference speed. Essahraui et al. reported that a full-precision (FP32) YOLO model achieved 99.5% mAP@0.5 on the NTHU Driver Drowsiness Detection (NTHU-DDD) dataset [2]. However, deploying such models on in-vehicle edge devices remains challenging because of limited computational resources. Lamaazi et al. addressed this issue by proposing an edge-based architecture that performs drowsiness detection locally, reducing network latency while preserving user privacy [3].

Despite these advantages, edge devices remain constrained by limited computational capability, memory, and power consumption, particularly under challenging conditions such as low-light environments. Abo-Zahhad et al. improved detection robustness by integrating Generative Adversarial Networks (GANs) with YOLOv5 and YOLOv8n to enhance low-quality images, although this approach substantially increases training complexity and computational overhead [4]. Recent studies have therefore emphasized the importance of lightweight deep learning models for efficient edge deployment [5].

Model optimization techniques such as quantization and pruning have become effective solutions for resource-constrained deployment. Quantization reduces the numerical precision of model weights and activations, such as from FP32 to FP16 or INT8, thereby reducing memory usage and inference time, while pruning removes redundant parameters with minimal impact on detection performance. Yang et al. demonstrated that structured pruning can reduce YOLO model parameters by up to 80% with only a minor loss in accuracy [6], whereas Naveen and Kounte reported that combining channel pruning with INT8 quantization improves inference speed by up to 22.72% without significantly degrading performance [7]. These studies demonstrate that model compression techniques can effectively improve computational efficiency while maintaining competitive detection performance for edge deployment.

YOLO11, introduced by Ultralytics, incorporates several architectural refinements to improve both detection accuracy and computational efficiency [8]. Its lightweight variant, YOLO11n, is specifically designed for resource-constrained environments, making it a suitable backbone for edge-based driver drowsiness detection. However, previous studies have generally evaluated quantization and pruning independently. Comprehensive comparisons of TensorRT quantization

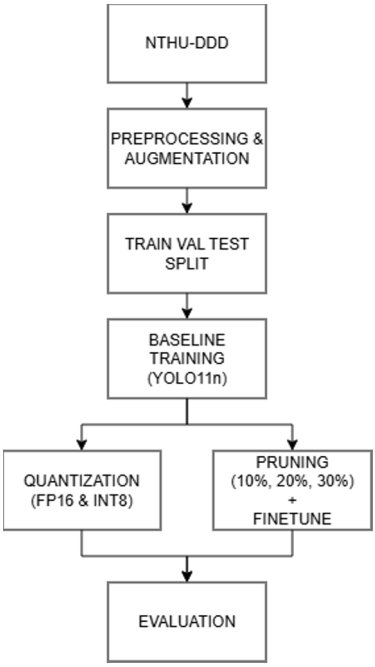


Fig. 1. Overall research workflow of the proposed YOLO11n optimization study.

and structured pruning on the YOLO11n architecture remain limited, particularly using the NTHU Driver Drowsiness Detection (NTHU-DDD) dataset.

Therefore, this study presents a systematic comparative evaluation of two YOLO11n optimization strategies: (1) TensorRT quantization (FP16 and INT8) and (2) unstructured magnitude pruning. Both strategies are evaluated on the NTHU-DDD dataset to analyze their effects on detection accuracy, computational efficiency, and model complexity. The findings provide practical guidance for selecting optimization strategies that enable accurate, lightweight, and real-time driver drowsiness detection systems for deployment on resource-constrained in-vehicle edge devices.

2 Proposed Method

This section describes the research steps taken to develop and evaluate a YOLO11n-based driver drowsiness detection model. The overall research workflow, encompassing all stages from dataset preparation through baseline model training to the application and evaluation of each optimization configuration, is illustrated in Figure 1.

2.1 Data Preparation

This study uses the NTHU Driver Drowsiness Detection (NTHU-DDD) dataset, one of the benchmark datasets widely used in research on driver drowsiness detection. This dataset consists of video recordings and facial images of drivers collected under various real-world conditions, including the use of regular glasses, the use of sunglasses, variations in lighting conditions, and variations in the driver's head orientation and pose. This diversity of conditions makes NTHU-DDD representative for testing a model's robustness against real-world scenarios.

Before the training process begins, all data first undergoes a preprocessing stage to ensure the quality and consistency of the model's input. Video data is extracted into individual frames in image format, then annotated manually or semi-automatically to mark the facial area along with the corresponding drowsiness condition. The annotation labels are subsequently converted into YOLO format, which is a text-based format using bounding box coordinates normalized to the image dimensions.

After the annotation process is complete, all images are resized to 224×224 pixels to maintain uniformity in input dimensions during training. This resolution was chosen to balance spatial information completeness and computational efficiency, particularly given the YOLO11n architecture's design for low computational requirements. The dataset is then divided into three partitions: the training set, the validation set, and the testing set, with respective ratios of 70%, 15%, and 15%. This division ensures that the model is evaluated on data it has never seen during training, so that the evaluation results objectively reflect the model's generalization ability. The step-by-step data preparation pipeline (extraction, annotation, resizing, and splitting) follows the corresponding stage of the overall workflow shown in Figure 1.

2.2 Hyperparameter Configuration

To ensure consistency and objectivity in cross-model comparisons, all experimental configurations in this study used identical hyperparameters. This approach aims to ensure that any differences in performance are solely due to the differences in the optimization strategies applied, rather than variations in training configurations. The hyperparameters used in all experiments are summarized in Table 1.

The AdamW optimizer was selected because it combines Adam's adaptive learning rate mechanism with decoupled weight decay regularization, resulting in more stable convergence and improved generalization performance. Compared with conventional Adam optimization, AdamW effectively reduces overfitting by separating weight decay from gradient updates. Furthermore, a learning rate of 1×10^{-4} was adopted to ensure smooth parameter updates during training while preventing unstable oscillations or convergence to poor local minima.

Table 1. Hyperparameter configuration used in all experiments.

Hyperparameters	**Value**
Input Size	224×224
Batch Size	32
Learning Rate	1×10^{-4}
Data Split	70 : 15 : 15
Optimizer	AdamW
Architecture	YOLO11n

2.3 Experimental Setup

TensorRT engines for the FP16 and INT8 configurations were compiled and benchmarked on the same hardware to ensure a fair comparison across precision formats.

2.4 Architecture

This research proposes a comparative study of four optimization strategies for the YOLO11n architecture to improve the efficiency of driver drowsiness detection models without significantly compromising detection performance.

YOLO11n + Quantization The first configuration applies post-training quantization via NVIDIA TensorRT to reduce memory consumption and accelerate inference. Quantization operates by mapping the continuous-valued floating-point representations of model weights and activations to lower-precision discrete formats. Two quantization schemes are investigated: FP16 (half-precision floating point) and INT8 (8-bit integer quantization).

FP16 quantization reduces per-parameter memory from 32 bits to 16 bits, achieving up to a $2\times$ reduction in model memory footprint with minimal numerical degradation due to the preserved dynamic range of the floating-point format. INT8 quantization reduces precision further to 8 bits per value, enabling up to a $4\times$ memory reduction and greater inference acceleration. However, because INT8 operates in a fixed-point domain with a substantially narrower dynamic range, a calibration step is required in which a representative subset of training data is passed through the network to compute optimal quantization scale factors for each layer's activation distribution. Crucially, neither quantization scheme modifies the underlying network topology, preserving the original model architecture while enabling hardware-accelerated inference through TensorRT-specific kernel fusion and graph optimization.

YOLO11n + Pruning The second configuration applies unstructured magnitude-based pruning to reduce the effective parameter count and associated computational overhead. In this approach, individual weight elements across all convolutional and fully connected layers are ranked by their L1 norm magnitude, and

Table 2. Performance comparison of YOLO11n under FP32, FP16, and INT8 TensorRT quantization.

Precision Format	mAP@50	mAP@50-95	Precision	Recall	FPS	Size (MB)
FP32 (Baseline)	0.9717	0.9068	0.9906	0.9255	32.44	5.17
FP16 (TRT)	0.9721	0.9078	0.8906	0.9328	35.91	6.02
INT8 (TRT)	0.9690	0.9047	0.8941	0.9223	34.30	4.80

those falling below a specified sparsity threshold are set to zero. This process is applied iteratively at sparsity levels of 10%, 20%, and 30% to examine the sensitivity of model performance to the degree of weight removal.

Because aggressive pruning can degrade feature extraction capacity, each pruned model undergoes a subsequent fine-tuning phase of 50 epochs, using the same learning rate and optimizer reported in Table 1, to allow the remaining active weights to compensate for the removed parameters and recover detection performance. Unlike structured pruning, which removes entire filters or channels and directly reduces architectural complexity, unstructured pruning introduces sparsity within the weight tensors while preserving the original layer structure, meaning that inference speedup is contingent on hardware support for sparse computation. Consequently, unstructured pruning does not reduce the total parameter count or model file size, as the zeroed weights remain physically present within the weight tensors and continue to occupy the same memory footprint as non-zero weights.

3 Result and Discussion

3.1 Evaluation of TensorRT Quantization Strategies: FP16 and INT8

The first phase of evaluation examines the impact of TensorRT-based model quantization on inference efficiency and detection accuracy. The results for the three quantization configurations, namely FP32 (baseline), FP16, and INT8, are summarized in Table 2 and illustrated in Figure 2. The quantization experiments were conducted using TensorRT on the same model weights to isolate the effect of numerical precision reduction from other variables. Reported FPS values in this subsection correspond to end-to-end per-image latency, including pre- and post-processing overhead, which differs from the pure model-inference throughput reported in Section 3.2; this distinction is discussed further in Section 4.

As shown in Figure 2 and Table 2, FP16 TensorRT quantization achieves the best overall detection performance, with the highest mAP@50 (0.9721), mAP@50-95 (0.9078), and Recall (0.9328), slightly outperforming the FP32 baseline. These improvements are attributed to TensorRT optimizations, including graph optimization, kernel fusion, and layer restructuring during engine compilation. However, Precision decreases from 0.9906 (FP32) to 0.8906, indicating

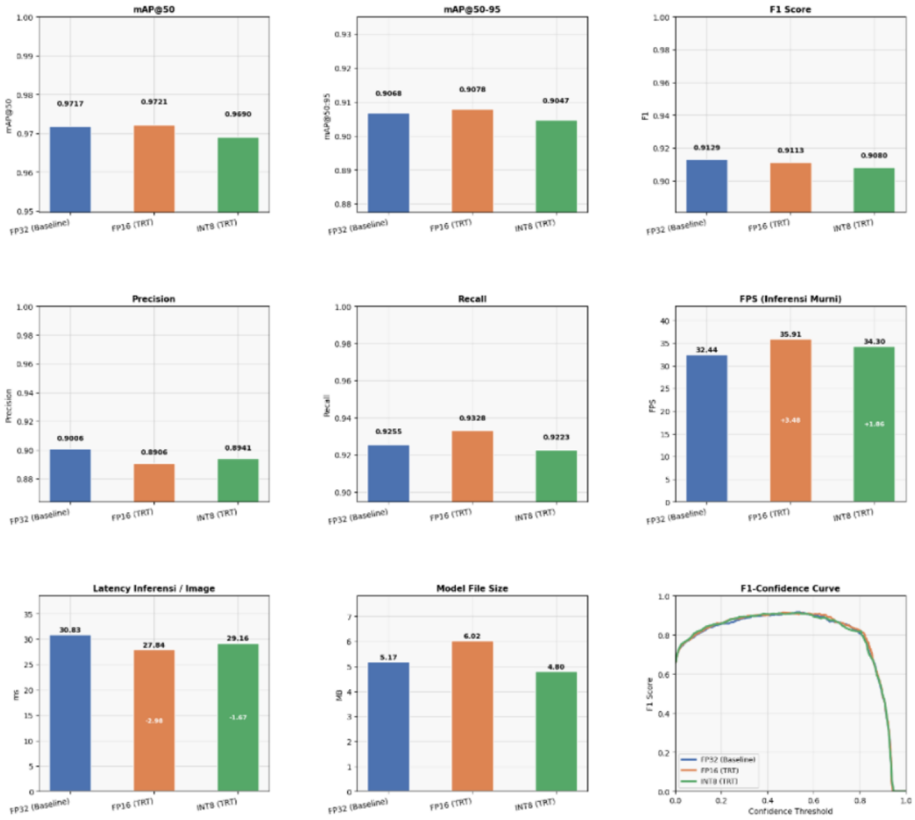


Fig. 2. Performance comparison of YOLO11n under FP32, FP16, and INT8 TensorRT quantization, showing detection metrics, inference speed, model size, and F1–Confidence curves.

an increase in false positive detections caused by reduced numerical precision. In terms of computational efficiency, FP16 delivers the highest inference speed of 35.91 FPS, reducing latency from 30.83 ms to 27.84 ms per image. The compiled TensorRT engine size increases slightly from 5.17 MB to 6.02 MB due to engine compilation artifacts.

The INT8 quantization model achieves an mAP@50 of 0.9690, mAP@50-95 of 0.9047, Precision of 0.8941, and Recall of 0.9223, with only a marginal reduction in detection performance compared with FP16. Nevertheless, it provides the smallest model size of 4.80 MB (7.2% smaller than FP32), an inference speed of 34.30 FPS, and a latency of 29.16 ms per image, making it the most memory-efficient configuration. As illustrated in Figure 2, the nearly identical F1-Confidence curves across all precision formats indicate that TensorRT quantization preserves detection robustness while significantly improving inference

Table 3. Performance comparison of YOLO11n under unstructured magnitude pruning at varying sparsity levels.

Model	Sparsity (%)	mAP@50	mAP@50–95	Precision	Recall	FPS
Baseline	0.0	0.9717	0.9068	0.9906	0.9255	66.49
Pruned 10%	10.0	0.9794	0.9094	0.9351	0.9237	66.60
Pruned 20%	20.0	0.9785	0.9059	0.9326	0.9299	66.95
Pruned 30%	30.0	0.8715	0.6865	0.6681	0.9272	68.45

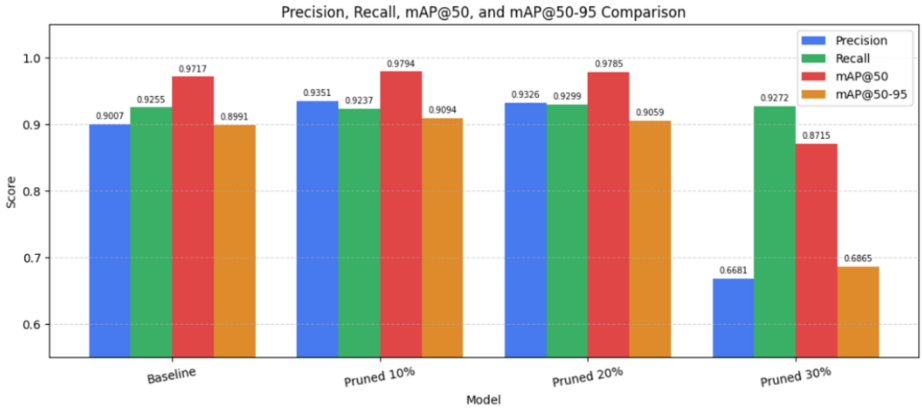


Fig. 3. Performance comparison of YOLO11n under unstructured magnitude pruning. The figure summarizes the effect of different sparsity levels on detection performance and inference speed.

efficiency, making both FP16 and INT8 suitable for real-time driver drowsiness detection on edge devices.

3.2 Evaluation of Unstructured Magnitude Pruning

The second phase of evaluation investigates the effect of unstructured magnitude-based pruning on the YOLO11n model across three sparsity levels: 10%, 20%, and 30%. In unstructured pruning, individual weights with the smallest L1 magnitudes are removed without altering the overall network architecture. Following each pruning operation, the model undergoes fine-tuning for 50 epochs to recover lost accuracy. The results are summarized in Table 3 and illustrated in Figure 3. Unlike Section 3.1, the FPS values reported here reflect pure model-inference throughput only, without pre- or post-processing overhead, which explains the higher absolute FPS values relative to Table 2 at comparable precision.

As shown in Table 3 and Figure 3, moderate unstructured pruning at 10% and 20% sparsity maintains competitive detection performance while preserving inference speed. The 10% pruned model achieves the highest mAP@50 (0.9794) and mAP@50–95 (0.9094), improving upon the baseline (0.9717 and 0.9068, respectively), while maintaining comparable Recall (0.9237) and inference speed

(66.60 FPS). Similarly, the 20% sparsity model records an mAP@50 of 0.9785, an mAP@50–95 of 0.9059, a slightly higher Recall of 0.9299, and an inference speed of 66.95 FPS. These results suggest that removing low-magnitude redundant weights produces a regularization effect that improves model generalization without significantly affecting detection accuracy. However, neither pruning level reduces model size or inference latency because zero-valued weights remain stored in dense tensors and continue to be processed by standard GPU operations.

In contrast, pruning at 30% sparsity causes a substantial decline in detection performance. The mAP@50 decreases to 0.8715, mAP@50–95 drops to 0.6865, and Precision falls sharply to 0.6681, although Recall remains relatively stable at 0.9272. The slight increase in inference speed to 68.45 FPS is negligible and does not reflect a meaningful computational benefit. This performance degradation indicates that 30% sparsity exceeds the redundancy capacity of the lightweight YOLO11n architecture, leading to irreversible loss of discriminative features that cannot be recovered through fine-tuning.

3.3 Discussion: Measurement Methodology and Study Limitations

The baseline FPS differs between Table 2 (32.44 FPS) and Table 3 (66.49 FPS) because the two experiments measure different things rather than reflecting an inconsistent model: Section 3.1 reports end-to-end latency (including pre- and post-processing, e.g., non-maximum suppression), while Section 3.2 reports pure model-inference throughput; future revisions will standardize this measurement across all strategies. Relative to prior work, this study differs from Yang et al. [6], who applied structured filter pruning that directly shrinks model size, by instead evaluating unstructured magnitude pruning, which preserves the architecture and depends on sparse-computation hardware support for any speed gain; it also differs from Naveen and Kounte [7], who combined channel pruning with INT8 quantization into one hybrid pipeline, by evaluating quantization and pruning independently and systematically on the same YOLO11n backbone and NTHU-DDD dataset, offering a controlled, strategy-by-strategy comparison that can inform future hybrid designs. A further limitation is that all inference measurements were obtained on the hardware described in Section 2.3 rather than on an in-vehicle edge device such as an NVIDIA Jetson or Raspberry Pi; although the FP16 and INT8 model sizes (6.02 MB and 4.80 MB) fit comfortably within typical embedded memory budgets, on-device throughput may differ due to compute capability, thermal throttling, and power constraints, so validating these strategies on physical edge hardware is prioritized as immediate future work.

4 Conclusion

This study presented a comprehensive comparative evaluation of YOLO11n optimization strategies for driver drowsiness detection on the NTHU-DDD dataset,

focusing on TensorRT quantization (FP16 and INT8) and unstructured magnitude pruning (10%, 20%, and 30% sparsity). The results demonstrate that each optimization strategy offers distinct trade-offs between detection performance and computational efficiency. FP16 TensorRT quantization provided the best balance between accuracy and real-time performance, achieving the highest inference speed of 35.91 FPS while maintaining detection accuracy comparable to the FP32 baseline. INT8 quantization produced the smallest model size (4.80 MB) with only a marginal reduction in detection performance, making it suitable for memory-constrained edge devices.

Moderate unstructured pruning at 10% and 20% sparsity improved model generalization, achieving mAP@50 values of 0.9794 and 0.9785, respectively, without affecting inference speed. However, pruning at 30% sparsity caused a substantial decline in detection performance, indicating that the lightweight YOLO11n architecture has limited redundancy for aggressive pruning. Overall, FP16 TensorRT quantization is the most practical optimization strategy for real-time edge deployment because it provides an effective balance between inference speed and detection accuracy. Future work should investigate structured pruning, hybrid optimization strategies that combine pruning and quantization, standardized cross-configuration FPS measurement, and validated deployment on embedded edge platforms, such as NVIDIA Jetson devices, to further improve efficiency and confirm real-world performance.

Acknowledgments. This research was supported by the Faculty of Computer Science, Universitas Brawijaya, Malang, Indonesia, through the organization and provision of this conference. The authors gratefully acknowledge the support and opportunity to present and disseminate this research.

References

1. World Health Organization: Global Status Report on Road Safety. WHO Press (2023)
2. Essahraui, S., Lamaakal, I., El Hamly, I., Maleh, Y., Ouahbi, I., El Makkaoui, K., Bouami, M.F., Pławiak, P., Alfarraj, O., Abd El-Latif, A.A.: Real-time driver drowsiness detection using facial analysis and machine learning techniques. Sensors **25**(3), 812 (2025). https://doi.org/10.3390/s25030812
3. Lamaazi, H., Alqassab, A., Fadul, R., Mizouni, R.: Smart Edge-Based Driver Drowsiness Detection in Mobile Crowdsourcing. IEEE Access **11**, 21863–21872 (2023). https://doi.org/10.1109/ACCESS.2023.3250834
4. Abo-Zahhad, M.M., Elghamrawy, S., Hefny, A.A., Elawady, Y.H.: Early drowsiness detection model in autonomous vehicles using GAN and YOLO integration. Neural Computing and Applications (2025). https://doi.org/10.1007/s00521-025-11676-7
5. Musa, A., Kakudi, H.A., Hassan, M., Hamada, M., Umar, U., Salisu, M.L.: Lightweight Deep Learning Models for Edge Devices: A Survey. International Journal of Computer Information Systems and Industrial Management Applications **17**, 189–206 (2025)
6. Yang, Y., et al.: Structured Filter Pruning for Efficient YOLO Object Detectors on Edge Hardware. Applied Intelligence **54**(4), 4312–4328 (2024)

7. Naveen, S., Kounte, M.R.: Optimized Convolutional Neural Network at the IoT Edge for Image Detection Using Pruning and Quantization. Multimedia Tools and Applications (2025). https://doi.org/10.1007/s11042-025-20811-y
8. Ultralytics: YOLO11 Architecture and Real-Time Object Detection Performance Docs. https://docs.ultralytics.com (2024)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

