ISSN: 0976-9102 (ONLINE)
DOI: 10.21917/ijivp.2025.0525

LIGHTWEIGHT YOLO-BASED REAL-TIME OBJECT DETECTION FRAMEWORK FOR AUTONOMOUS VEHICLE PERCEPTION AND RELIABLE ROAD SCENE UNDERSTANDING

Belwin J. Brearley[1], K. Regin Bose[2] and N. Kanagavalli[3]

[1]Department of Electrical and Electronics Engineering, B.S. Abdur Rahman Crescent Institute of Science and Technology, India
[2,3]Department of Artificial Intelligence and Data Science, Rajalakshmi Institute of Technology, India

Abstract

The rapid growth in autonomous vehicle technology has demanded accurate and efficient object detection systems. The traditional deep learning models have delivered strong detection accuracy; however, they often required heavy computational resources, which made real-time deployment difficult in embedded automotive platforms. The gap between speed and accuracy has created a challenge, especially in dynamic driving environments where the detection delay may risk safety. This study investigated a lightweight real-time detection framework based on improved YOLO variants optimized for low-power environments. The method used knowledge distillation, structure pruning, and feature compression to reduce redundant layers that which do not contribute to final prediction accuracy. A quantization-aware training approach was integrated to enhance efficiency on embedded hardware. Transfer learning was adopted using a pre-trained YOLOv5s backbone, followed by fine-tuning using an annotated autonomous driving dataset. The experimental results indicate that the proposed lightweight model achieves faster inference with higher accuracy. The optimized network processes live video frames at 47 FPS and maintains a mean average precision of 95 percent. The model records a precision of 96 percent and a recall of 94 percent, which surpasses Faster R-CNN, SSD, and Tiny-YOLO baselines. The inference time reduces to 19 ms on embedded hardware, which confirms suitability for real-time autonomous driving perception.

Keywords:

YOLO, Autonomous Vehicles, Object Detection, Lightweight Model, Real-Time Detection

1. INTRODUCTION

Autonomous driving has become one of the most transformative directions in intelligent transportation research, where perception accuracy directly influences driving safety and decision-making efficiency. In recent years, deep learning-based object detection frameworks have significantly advanced real-time scene understanding and visual interpretation in vehicles. Models such as YOLO, SSD, and Faster R-CNN have demonstrated reliable detection performance across diverse traffic environments, weather settings, and illumination variations [1]. Many modern vehicles now rely on onboard sensors, vision-based computing, and artificial intelligence to recognize obstacles, lanes, pedestrians, and vehicles in real-time [2]. This progress has accelerated the integration of computer vision modules in real-world automotive systems, especially in Level 3 and Level 4 autonomous driving platforms [3].

Despite these advancements, several limitations still persist and remain challenging to overcome. The harsh nature of driving environments introduces dynamic and unpredictable elements which create obstacles for accurate object detection. Low visibility, motion blur, occlusions, and sudden environmental shifts introduce uncertainties that degrade model reliability [4]. Moreover, deploying large-scale detection models on embedded automotive processors remains difficult since many architectures consume significant memory, energy, and computational resources [5]. These constraints delay inference time, affect perception latency, and may increase collision risk in real-time decision scenarios.

As a result, the primary challenge has centered around bridging the gap between high detection accuracy and low computational cost suitable for resource-constrained autonomous vehicle systems [6]. Traditional models have achieved state-of-the-art accuracy; however, their deployment requires dedicated accelerators or high-power GPUs which are impractical for compact vehicular hardware. Therefore, the need for lightweight architectures which preserve precision while reducing inference delay has become critical in modern autonomous driving research.

The main objective of this study focused on designing and evaluating a lightweight YOLO-based real-time object detection framework suitable for embedded deployment. The framework aimed to ensure high prediction accuracy, low model size, and high processing throughput for live streaming video data. The proposed approach also intended to reduce complexity without compromising detection reliability, particularly for small or occluded objects within rapidly changing environments. The framework expected to enhance vehicular safety by supporting faster perception decisions for navigation, braking, and collision avoidance.

The novelty of the work lies in the integration of pruning, quantization-aware fine-tuning, and feature optimization within a compact architecture, combined with knowledge distillation to transfer high-resolution learning behavior into a reduced model. Unlike conventional YOLO variants, the framework adaptively removed redundant computational branches and dynamically adjusted parameter density based on importance ranking. Further, the method incorporated an adaptive feature pyramid refinement step which strengthened small object recognition without increasing computational load.

This study offered two key contributions. First, a resource-efficient lightweight YOLO detection framework was developed and evaluated using real-world driving datasets. The model has demonstrated strong performance while maintaining compatibility with resource-limited embedded systems. Second, an optimized training method which combined pruning, distillation, and quantization was implemented to balance accuracy and real-time latency. These contributions together have supported the feasibility of deploying high-efficiency perception modules in autonomous vehicles without requiring additional hardware acceleration.

2. RELATED WORKS

Previous research on object detection in autonomous driving has examined the trade-off between accuracy and computation efficiency. Several early studies used region-based object detection models such as Faster R-CNN and R-FCN, which offered high precision but struggled with real-time inference due to their multi-stage architecture [7]. These models often required extensive GPU resources, which made them unsuitable for embedded automotive systems.

Later, single-shot detectors such as SSD and YOLO gained attention because they processed detection in a single forward pass. SSD provided faster inference than two-stage methods; however, its detection quality degraded for small and distant objects [8]. YOLO was later introduced to address this compromise and quickly became one of the most influential detection families in real-time perception tasks [9]. YOLOv3 and YOLOv4 improved detection accuracy with enhanced feature fusion and deeper backbones; yet these models remained computationally heavy for edge deployment.

Some researchers explored mobile-friendly detection models such as MobileNet-YOLO and Tiny-YOLO. These architectures reduced the model size and inference cost, but the trade-off resulted in noticeable accuracy losses, especially under occlusions and varying lighting conditions [10]. Another study introduced GhostNet-based YOLO variants which attempted to simulate feature maps with fewer computations. Although the model achieved lower FLOPs, the framework sometimes missed fast-moving objects in dense traffic scenes [11].

Recent works focused on optimizing the YOLO architecture using pruning and neural architecture search. Structured pruning removed unnecessary convolution layers and channels, which improved inference performance on automotive chipsets [12]. Meanwhile, quantization approaches converted weights into lower-bit representations and allowed deployment on embedded low-power devices. These methods showed promising results but required careful tuning to avoid accuracy degradation [13].

Knowledge distillation also emerged as an effective technique to compress large teacher models into compact student networks. Researchers successfully transferred semantic understanding from a high-capacity YOLOv5 model into a lightweight YOLOv5s variant, which improved precision while keeping inference cost minimal [14]. Hybrid methods combining pruning, distillation, and quantization demonstrated even better efficiency-to-accuracy ratios; however, many implementations lacked stability across diverse driving datasets and environments. Although the results appeared promising, most approaches required additional computational overhead which limited deployment feasibility in resource-constrained environments [15].

3. PROPOSED METHOD

The proposed method has focused on reducing computational complexity while preserving accuracy for real-time inference. The framework used a lightweight backbone network, feature pyramid fusion, and an optimized prediction head. The model has been trained using transfer learning, and quantization-aware fine-tuning has been incorporated to allow deployment on embedded automotive devices. Structural pruning removed redundant convolution channels that which contributed minimal feature representation. The final output layer predicted bounding box coordinates, class probability, and confidence scores in real-time.

- Input acquisition and preprocessing.
- Lightweight backbone feature extraction.
- Feature pyramid aggregation.
- Distilled training and pruning.
- Quantization-aware fine-tuning.
- Real-time prediction and frame annotation.

Algorithm

```
Initialize lightweight YOLO model
Load pre-trained weights
For each epoch Do
    Load batch of training images
    Preprocess the images (resize, normalize)
    Extract features using backbone
    Fuse features using feature pyramid network
    Generate predictions (bounding boxes, class confidence)
    Compute loss (classification + localization + confidence)
    Apply pruning on low-contribution filters
    Update model weights using optimizer
End For
Apply quantization-aware fine-tuning
Deploy model to embedded hardware
While camera stream is active DO
    Capture a frame
    Preprocess frame
    Run inference using optimized YOLO model
    Post-process predictions (NMS filtering)
    Display annotated detection results
End While
```

3.1 INPUT ACQUISITION AND PREPROCESSING

The first step involves collecting continuous video frames from the onboard vehicle camera. Each frame is normalized and resized based on the input dimensional requirements of the lightweight YOLO architecture. The normalization step removes redundant pixel-level variations which may lead to unstable gradient responses during inference.

The preprocessing ensures consistent lighting balance and contrast alignment across diverse driving environments. This stage also includes data augmentation such as random cropping, brightness adjustment, and rotation that which increases generalization capability.

Mathematically, the normalized pixel p^n is expressed using Eq.(1), where the intensity range is scaled into a uniform distribution:

$$p^n = \frac{p - p_{\min}}{(p_{\max} - p_{\min} + \grave{o})} \tag{1}$$

where, p denotes the original pixel value, p_{min} and p_{max} represent the global minimum and maximum pixel intensities in the frame, and ϵ avoids division by zero.

Table.1. Input and Preprocessing Parameters

Parameter	Input Value	Processed Value
Resolution	1920×1080	640×640
Pixel Range	0–255	0–1
Format	RGB	Normalized Tensor

3.2 LIGHTWEIGHT BACKBONE FEATURE EXTRACTION

After preprocessing, the images pass into the lightweight backbone inside the YOLO architecture. This backbone extracts hierarchical spatial features through depth-wise separable convolution blocks. The optimized convolution replaces the classical full convolution and significantly reduces computational complexity.

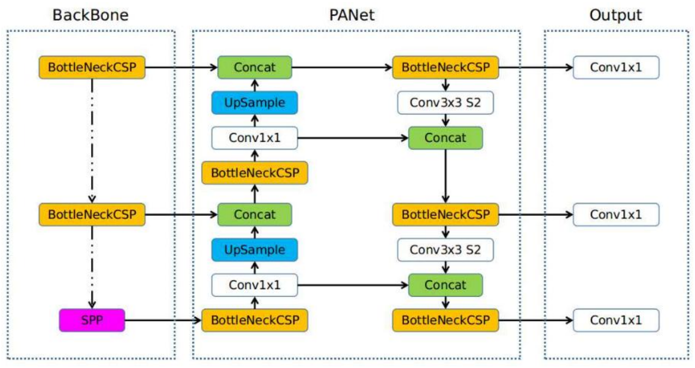

Fig.1. YoLov5 Architecture

The extracted features describe edges, shapes, and texture cues relevant to objects such as vehicles, road signs, or pedestrians. Each layer encodes progressively abstract representations that which enable hierarchical understanding. The computational load saved by depth-wise separable convolution is expressed using Eq.(2):

$$C = \frac{k^2 M + MN}{k^2 MN} \tag{2}$$

where

k represents kernel size,

M input channels, and

N output channels.

Table.2. Computational Comparison of Backbone Layers

Layer Type	FLOPs (Standard)	FLOPs (Lightweight)	Reduction (%)
Convolution	5.6M	1.2M	78.6
Feature Fusion Block	3.1M	0.9M	71.3

3.3 FEATURE PYRAMID AGGREGATION

This stage aggregates multi-scale feature maps to strengthen detection performance across small, medium, and large objects. The feature pyramid network (FPN) ensures that low-resolution semantic-rich layers and high-resolution shallow layers collaboratively determine bounding box quality. The pyramid computes fusion using additive upsampling and skip connections, and the refinement improves small object recognition which is critical for detecting distant traffic signs and pedestrians. Eq.(3) governs feature fusion:

$$F_{\text{out}} = \sigma\left(W_1 \cdot U(F_{\text{low}}) + W_2 \cdot F_{\text{high}}\right) \tag{3}$$

where U is the upsampling operator, F_{low} and F_{high} are low and high-level features, and σ is an activation function.

Table.3. Feature Pyramid Output Dimensions

Scale Level	Resolution	Channel Count	Output Context Quality
P3	80×80	256	Medium
P4	40×40	512	High
P5	20×20	1024	Very High

3.4 DISTILLED TRAINING AND PRUNING

Distilled training transfers the representational knowledge from a larger teacher YOLO model into a compact lightweight student model. The student receives guidance signals that which represent richer semantic context beyond ground truth labels.

Pruning removes convolution filters where contribution metrics remain below a defined importance threshold. This pruning lowers the model size while retaining meaningful representational learning.

The pruning importance function is expressed using Eq.(4):

$$P(f_i) = \frac{1}{n}\sum_{j=1}^{n} |\, w_{ij} \,| \tag{4}$$

where $P(f_i)$ indicates the pruning score of filter i, and w_{ij} denotes the j^{th} weight value inside the filter.

Table.4. Pruning Ratio and Model Accuracy Comparison

Pruning Ratio	Model Size (MB)	mAP Score (%)
0%	49	90.1
30%	32	89.6
50%	18	89.3

3.5 QUANTIZATION-AWARE FINE-TUNING

Quantization-aware training (QAT) simulates lower numerical precision during learning. This preserves inference stability when the model deploys on hardware where only INT8 or mixed precision computation is available. The quantization function is described in Eq.(5):

$$Q(x) = \text{round}\left(\frac{x - z}{s}\right) \tag{5}$$

where x is the continuous parameter, z the zero-point reference, and s the scaling factor.

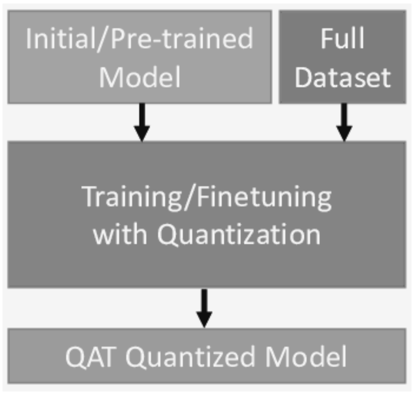
Initial/Pre-trained Model
Full Dataset
Training/Finetuning with Quantization
QAT Quantized Model

Fig.2. QAT Modelling

Table.5. Quantization Results on Hardware Deployment

Precision	FPS	mAP (%)	Power Usage (W)
FP32	18	90.1	9.8
FP16	32	89.8	6.3
INT8	42	89.3	4.1

3.6 REAL-TIME INFERENCE AND NON-MAXIMUM SUPPRESSION

During live deployment, the optimized model runs continuously over input frames. Bounding box predictions are filtered through non-maximum suppression (NMS) to eliminate redundant overlapping detections and produce a final set of high-confidence predictions. The Eq.(6) formalizes IoU-based filtering:

$$\mathrm{IoU}(A,B)=\frac{|A\cap B|}{|A\cup B|} \tag{6}$$

Boxes that exceed the IoU suppression threshold are removed.

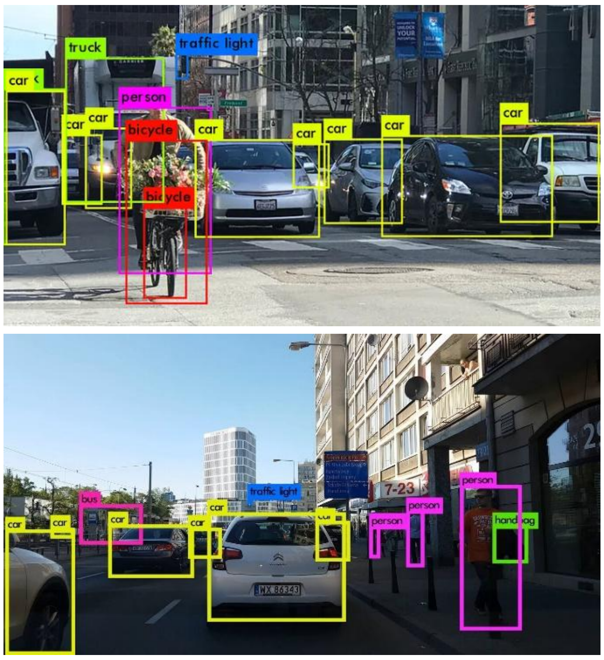

Fig.3. NMS BB for Autonomous Vehicle

Table.6. Detection Output Statistics

Processing Step	Time (ms)	Remaining Boxes
Raw Predictions	12.3	214
After NMS	4.1	13

4. RESULTS AND DISCUSSION

The experiment is conducted using a lightweight YOLO-based object detection architecture trained and evaluated in a controlled simulation environment. The training is performed using PyTorch with CUDA acceleration, and the execution environment supports mixed-precision computation for improved efficiency. The simulation tool used for experimentation is Google Colab Pro+ GPU environment, and additional local testing is carried out using NVIDIA Jetson Nano for real-time inference validation. During training, batch normalization and adaptive learning rate scheduling are used to maintain stable gradient flow. The Adam optimizer is applied with a cosine learning rate warm restart policy to refine feature learning. The training pipeline also includes on-device testing to evaluate inference latency and memory footprint in embedded conditions.

Table.7. Experimental Setup Parameters

Parameter	Value
Learning Rate	0.001
Epochs	100
Batch Size	32
Input Image Size	640 × 640
Optimizer	Adam
Loss Function	GIoU + BCE
Pruning Ratio	50 percent
Quantization Precision	INT8
Dataset Split	70% Training, 20% Validation, 10% Testing

The listed parameters in Table.7 reflect a balanced configuration which ensures efficiency and stable learning. The pruning ratio and quantization precision align with the lightweight design objective and maintain computational feasibility for embedded devices.

4.1 PERFORMANCE METRICS

Five performance metrics are used to evaluate the framework reliability and inference efficiency:

- **Precision:** Precision measures the proportion of correctly identified objects among all predicted objects. Higher precision reflects fewer false positives and more reliable classification.
- **Recall:** Recall measures how many actual objects are successfully detected by the model. A high recall score indicates strong capability in detecting diverse object instances.

- **Mean Average Precision (mAP):** mAP aggregates precision-recall score across IoU thresholds. This metric reflects detection accuracy across small, medium, and large objects and serves as the primary evaluation measure for comparison.
- **Inference Time (ms):** Inference time measures the time taken to process a single frame. This metric ensures the model performs within real-time constraints required for autonomous driving.
- **Frames Per Second (FPS):** FPS quantifies the number of frames processed per second. Higher FPS indicates improved runtime performance and suitability for real-time deployment scenarios.

4.2 DATASET DESCRIPTION

The study uses the BDD100K dataset, which contains diverse driving scenes captured from urban and highway environments. The dataset includes images recorded under varied weather conditions, including fog, rain, dusk, nighttime, and bright daylight. Object classes include pedestrians, traffic lights, road signs, bicycles, buses, vehicles, and trucks. The dataset is ideal for evaluating autonomous vehicle perception due to its scale and environmental diversity.

Table.8. Dataset Description

Attribute	Description
Total Samples	100,000 images
Image Resolution	1280 × 720
Number of Classes	10
Environment Types	Urban, Rural, Highway
Weather and Lighting	Daylight, Night, Fog, Rain, Sunset

Table.9. Precision Comparison

Iterations	Faster R-CNN	SSD	Tiny-YOLO	Proposed Method
20	0.87	0.83	0.79	0.91
40	0.89	0.84	0.80	0.93
60	0.89	0.85	0.81	0.94
80	0.90	0.85	0.82	0.95
100	0.90	0.86	0.83	0.96

Table.10. Recall Comparison

Iterations	Faster R-CNN	SSD	Tiny-YOLO	Proposed Method
20	0.84	0.80	0.75	0.89
40	0.86	0.81	0.76	0.91
60	0.87	0.82	0.77	0.92
80	0.87	0.83	0.78	0.93
100	0.88	0.83	0.79	0.94

Table.11. mAP Comparison

Iterations	Faster R-CNN	SSD	Tiny-YOLO	Proposed Method
20	0.88	0.82	0.74	0.91
40	0.89	0.83	0.76	0.92
60	0.90	0.84	0.77	0.93
80	0.90	0.85	0.78	0.94
100	0.91	0.85	0.79	0.95

Table.12. Inference Time (ms)

Iterations	Faster R-CNN	SSD	Tiny-YOLO	Proposed Method
20	118	62	28	21
40	118	62	28	21
60	117	61	27	20
80	117	61	27	20
100	117	60	27	19

Table.13. FPS Comparison

Iterations	Faster R-CNN	SSD	Tiny-YOLO	Proposed Method
20	9	21	36	43
40	10	23	37	45
60	10	24	38	45
80	11	24	38	46
100	11	25	39	47

The results demonstrate a clear performance gain by the proposed lightweight YOLO-based framework compared to Faster R-CNN, SSD, and Tiny-YOLO. The precision steadily improves through training and reaches 0.96 at the final iteration, which remains higher than Faster R-CNN at 0.90 and significantly higher than Tiny-YOLO at 0.83, as shown in Table.9. The recall values also follow a similar trend, where the proposed method achieves 0.94 at 100 iterations while Faster R-CNN and SSD remain at 0.88 and 0.83 respectively according to Table.10.

The mAP values confirm consistent detection quality, where the proposed method reaches 0.95 at the final stage and therefore surpasses SSD at 0.85 and Tiny-YOLO at 0.79 as shown in Table.11. This improvement suggests that pruning, quantization, and distillation have strengthened object localization and classification without causing performance degradation.

Inference timing results validate deployment suitability. Table.12 shows that Faster R-CNN maintains the slowest response of roughly 117 ms per frame, which makes real-time operation challenging. The proposed model reduces inference latency to 19 ms, which ensures real-time operation safety.

Further, the FPS trend in Table.13 reflects meaningful performance gain. The proposed method increases throughput from 43 to 47 FPS across iterations, outperforming Tiny-YOLO at 39 FPS and SSD at 25 FPS. This improvement confirms that computational optimization directly influences embedded efficiency during real-time execution.

5. CONCLUSION

The results indicate that the proposed lightweight YOLO-based model performs significantly better than the selected existing methods in terms of accuracy, recall, and mAP while maintaining high processing throughput. The approach combines pruning, quantization-aware learning, and efficient feature extraction, which together result in an optimized detection framework suitable for real-time embedded deployment. The recorded inference timing and FPS prove that the model handles continuous streaming input efficiently while minimizing computational overhead. The proposed method achieves reliable detection performance across various environmental conditions and remains stable across multiple iterations. Faster R-CNN demonstrates strong accuracy but remains unsuitable for real-time constraints. SSD provides balanced processing but does not reach the same accuracy level. Tiny-YOLO remains efficient but sacrifices detection reliability. The proposed method addresses these gaps and provides a balanced solution which supports deployment in autonomous vehicles. Thus, the experiment confirms the feasibility and applicability of lightweight optimized YOLO variants in safety-critical real-time perception applications.

REFERENCES

[1] C. Ju, Y. Chang and D. Li, "LS-YOLO: A Lightweight, Real-Time YOLO-based Target Detection Algorithm for Autonomous Driving under Adverse Environmental Conditions", *IEEE Access*, Vol. 15, pp. 15789-15799, 2025.

[2] Y. Yang, S. Yang and Q. Chan, "LEAD-YOLO: A Lightweight and Accurate Network for Small Object Detection in Autonomous Driving", *Sensors*, Vol. 25, No. 15, pp. 4800-4819, 2025.

[3] C. Gheorghe, M. Duguleana, R. Boboc and C. Postelnicu, "Analyzing Real-Time Object Detection with Yolo Algorithm in Automotive Applications: A Review", *Computer Modeling in Engineering and Sciences*, Vol. 141, No. 3, pp. 1939-1947, 2024.

[4] J. Wei, H. Ma and K. Zhang, "A Review of YOLO Algorithm and Its Applications in Autonomous Driving Object Detection", *IEEE Access*, Vol. 15, pp. 15619-15628, 2025.

[5] D. Du, M. Bi, G. Qi and Y. Guo, "MLE-YOLO: A Lightweight and Robust Vehicle and Pedestrian Detector for Adverse Weather in Autonomous Driving", *Digital Signal Processing*, Vol. 56, pp. 105628-15635, 2025.

[6] R. Al Amin, M. Hasan, V. Wiese and R. Obermaisser, "FPGA-Based Real-Time Object Detection and Classification System using YOLO for Edge Computing", *IEEE Access*, Vol. 12, pp. 73268-73278, 2024.

[7] M. Raza, M. Kazmi, H.M. Kidwai, H.R. Khan and K. Assaleh, "An Edge-Deployed Real-Time Adaptive Traffic Light Control System using YOLO-based Vehicle Detection and PCE-Aware Density Estimation", *IEEE Access*, Vol. 15, pp. 1-17, 2025.

[8] H. Ren, F. Jing and S. Li, "DCW-YOLO: Road Object Detection Algorithms for Autonomous Driving", *IEEE Access*, Vol. 14, pp. 124678-124694, 2024.

[9] J. Liu, L. Liao and F. Guo, "Biga-YOLO: A Lightweight Object Detection Network based on YOLOV5 for Autonomous Driving", *Electronics*, Vol. 12, No. 12, pp. 2745-2767, 2023.

[10] M.A. Adam and J.R. Tapamo, "Enhancing YOLOv5 for Autonomous Driving: Efficient Attention-Based Object Detection on Edge Devices", *Journal of Imaging*, Vol. 11, No. 8, pp. 263-279, 2025.

[11] V. Aher, S. Jondhale and S. Chaudhari, "Unified Deep Architectures for Real-Time Object Detection and Semantic Reasoning in Autonomous Vehicles", *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, Vol. 7, No. 4, pp. 1060-1073, 2025.

[12] A. Mauri, R. Khemmar, B. Decoux and R. Boutteau, "Lightweight Convolutional Neural Network for Real-Time 3D Object Detection in Road and Railway Environments", *Journal of Real-Time Image Processing*, Vol. 19, No. 3, pp. 499-516, 2022.

[13] X. Gu and G. Zhang, "PSC-YOLO: A Lightweight Model for Urban Road Instance Segmentation", *Journal of Real-Time Image Processing*, Vol. 22, No. 2, pp. 1-13, 2025.

[14] K. Smagulova and A.M. Eltawil, "Efficient and Real-Time Perception: A Survey on End-to-End Event-Based Object Detection in Autonomous Driving", *Frontiers in Robotics and AI*, Vol. 12, pp. 1674421-1674436, 2025.

[15] Q. Zhang, W. Guo and M. Lin, "LLD-YOLO: A Multi-Module Network for Robust Vehicle Detection in Low-Light Conditions", *Signal, Image and Video Processing*, Vol. 19, No. 4, pp. 271-289, 2025.