Edge-Based Deep Learning for Traffic Object Detection in Driving Scenarios

Hanif Rahmat
Department of Computer Science and Electronics, Universitas Gadjah Mada, Sleman, Indonesia
hanifrahmat123@gmail.com

Suprapto
Department of Computer Science and Electronics, Universitas Gadjah Mada, Sleman, Indonesia
sprapto@ugm.ac.id (corresponding author)

Moh Edi Wibowo
Department of Computer Science and Electronics, Universitas Gadjah Mada, Sleman, Indonesia
mediw@ugm.ac.id

Received: 26 January 2026 | Revised: 7 April 2026 | Accepted: 17 April 2026

 DOI: https://doi.org/10.48084/etasr.17760

ABSTRACT

Object detection is an important task in autonomous driving. Accuracy and processing speed often become a trade-off in edge deep learning-based object detection implementations. This study aimed to comparatively analyze the performance of deep learning-based object detection methods at the edge in detecting traffic-related objects in driving scenarios. This study contributes to the field by implementing fine-tuning methods on a custom dataset, on driving scenarios in Yogyakarta, Indonesia, and deploying them onto a resource-constrained edge computing device. PyTorch is used as the deep learning framework, with TorchVision as the main library, and Open Neural Network Exchange (ONNX) is used to convert PyTorch models into a standardized graph for edge implementation. The experimental results show that the two-stage detector Faster R-CNN ResNet50 with an input size of 800 outperforms other methods, achieving the best mAP of 37.6% and mAPS of 21.1%, but its inference time was the longest, reaching 78.89 s. In contrast, Faster R-CNN MobileNetV3-Large with an input size of 320 achieved the fastest inference time (0.58 s) with significantly lower mAP, mAP0.5, and mAP0.75 (11.2%, 19.5%, 11.9%, respectively). Conditional DETR achieved a moderate inference time (17.31 s) and considerable mAP (32.7%). FCOS with an input size of 800 had a higher mAP than Conditional DETR (34.8% vs. 32.7%) and an inference time twice as fast as Faster R-CNN ResNet50 (38.92 vs. 78.89 s). Therefore, Conditional DETR and FCOS are preferable for resource-constrained edge computing implementations.

Keywords-deep learning; object detection; edge computing; traffic; driving scenarios; fine-tuning

I. INTRODUCTION

Object detection plays a crucial role in autonomous driving, as it enables vehicles to perceive and understand their surrounding environment. Accurate detection of traffic-related objects in driving scenarios, such as vehicles, pedestrians, and traffic signs, is important for real-world applications such as advanced driver assistance systems and autonomous driving. Object detection performance affects driving safety, navigation efficiency, and real-time decision-making in complex traffic environments [1]. In recent years, object detection has advanced significantly with deep learning approaches, which have demonstrated superior accuracy compared to traditional methods. Deep learning-based object detectors can be broadly categorized into two-stage methods, such as Faster R-CNN, and one-stage methods, including SSD and RetinaNet [2-4]. More recently, Transformer-based detectors have emerged, pioneered by DETR, which has streamlined the detection pipeline, removing hand-designed components such as non-maximum suppression and anchor generation. DETR has been followed by its improved variants, including Conditional DETR [5, 6], Deformable DETR [7], and SAM-DETR [8].

Since object detection in driving scenarios requires real-time processing, the deployment of the models is closely related to edge computing, where inference is performed directly on edge devices. However, edge devices typically have limited computational resources, such as a processor and memory. As a result, it is essential to investigate how different object detection methods perform under such limitations, particularly in terms of accuracy or precision and inference time.

In machine learning, transfer learning is a technique that uses learned knowledge from a source domain to solve a target

domain problem. The model parameters from a pretrained model can be used to address a new task without the need to train the model from scratch. Thus, training time can be reduced significantly. Transfer learning is highly preferable when the dataset size is limited, or for real-world artificial intelligence applications [9]. Fine-tuning is important in transfer learning. Fine-tuning leverages models pre-trained on a dataset, usually larger in size, and adapts them to a different specific task or domain [10, 11]. For example, in [12], pre-trained models of Faster R-CNN and SSD were fine-tuned for pedestrian detection. With only 700 data samples, a high performance was achieved, with precisions of 96.6% and 92% for Faster R-CNN and SSD, respectively. In [13], pre-trained Faster R-CNN, SSD, and YOLOv3 models were fine-tuned on 417 training data samples to detect abnormalities in melon leaves.

This study focuses on traffic-related object detection in driving scenarios using transfer learning and fine-tuning. To support the development of autonomous driving in Yogyakarta, Yogyakarta traffic was considered a case study, which presents a particularly challenging setting for object detection due to its diverse and dense traffic conditions. The coexistence of various vehicle types, frequent occlusions, and the presence of small objects such as motorcycles, bicycles, and street-side obstacles creates a highly complex visual environment. These characteristics make Yogyakarta an important case study for developing and evaluating object detection systems that are better adapted to real-world driving conditions, especially for applications such as autonomous or semi-autonomous driving in developing urban areas.

This study presents a comparative analysis of object detection performance on a Raspberry Pi-based edge device for detecting traffic-related objects in driving scenarios. Raspberry Pi was selected due to its low cost and practical relevance for real-world edge deployment, despite its limited computational capability. Through this analysis, the study aims to provide insights into the trade-offs between detection precision and inference speed when applying state-of-the-art object detection models to resource-constrained edge devices in real driving scenarios.

This work is an extension of [14]. The earlier study focused on improving the detection performance of SAM-DETR [14]; however, this study focuses on benchmarking several object detection models, such as Faster R-CNN, SSD, and RetinaNet, and their implementation on a resource-constrained edge device using the same primary dataset. This allows for a comprehensive understanding of how established models perform on this specific dataset within resource-constrained environments.

II. METHODOLOGY

A. Dataset

The dataset employed in this study is identical to the primary dataset described in [14] and comprises 3,144 images that capture driving scenarios in Yogyakarta, Indonesia. Each scenario was captured using a dashboard camera mounted in front of a car. First, driving scenario videos were downloaded from the YouTube channel "Roadscape ID," which provides driving videos in various places, times, and conditions. The downloaded videos are in different time backgrounds, including morning, afternoon, evening, and night. This step resulted in nine videos at a resolution of 1080p. Second, images were extracted from every frame of i −th second, where $i = 5, 10, 15, \ldots$. From this phase, 3,144 images were obtained, each at a resolution of 1920×1080 pixels [14].

The images were then annotated by assigning bounding boxes and related classes to traffic-related objects in each image. The annotation was done using the tool "labelme" and then exported to the COCO JSON annotation format. From this annotation step, a total of 18 object categories were produced, as presented in Table I. Some typical terms were let in Bahasa Indonesia, i.e., gerobak, becak, tukang-becak, and delman. A becak is a three-wheeled pedicab used for short-distance passenger transportation, while tukang becak is a becak driver. However, in this case, the class tukang-becak is defined as a becak that is being actively ridden or operated by a becak driver. This rule is also applied to the motorcyclist and bicyclist classes; however, the bicycle class is excluded due to an insufficient number of instances (fewer than 100). A delman is a traditional horse-drawn carriage with two or four wheels, used for passenger transport. A gerobak is a traditional handcart or pushcart used for purposes such as goods transportation, mobile vending, general-purpose hauling tools, etc. [14]. Figure 1 depicts examples of objects of these classes.

Finally, the annotated dataset was split into three subsets, i.e., training, validation, and testing sets with proportions of 70%, 15%, and 15%, respectively. Table I presents the distribution of the dataset.

B. Training Details

This study employed fine-tuning for the detection of traffic-related objects in driving scenarios. Pre-trained models were used, including state-of-the-art models like Faster R-CNN, SSD, SSD-Lite, RetinaNet, FCOS, DETR, and Conditional DETR. These methods were selected based on their availability in standard deep learning libraries and their compatibility with the Open Neural Network Exchange (ONNX) graph format, which facilitates optimization for edge deployment. PyTorch was used as the deep learning framework with TorchVision as the main library. In this study, all the parameters' weights are updated during training on the custom dataset. The models were trained with an Nvidia A100 GPU under a 24-epoch training scheme. The batch size was set to four. For Faster R-CNN, SSD, SSD-Lite, RetinaNet, and FCOS, the learning rate was set to 1×10^{-3}. Stochastic Gradient Descent (SGD) was used as the optimizer. The momentum and weight decay were set to 0.9 and 5×10^{-4}, respectively. Random horizontal flip was applied for augmentation during training.

For DETR and Conditional DETR, the learning rate was set to 1×10^{-4} for the Transformer encoder-decoder and 1×10^{-5} for the backbone. In addition, AdamW was used with a weight decay of 1×10^{-4}. The number of object queries N and the dimension of the Feed-Forward Network (FFN) were set to 300 and 2048, respectively. Both models used sine positional embeddings. Standard augmentation techniques were used as in the original implementation, including horizontal flip, random

crop, and random resize with the longest side at most 1333 pixels and the shortest side at least 480 pixels and at most 800 pixels. For brevity, the 480–800-pixel range for the shortest side is simply denoted as 480 pixels. Table II presents details on the backbones and input sizes.

Fig. 1. Object examples: (a) becak, (b) tukang-becak, (c) delman, and (d) gerobak.

TABLE I. CLASS DISTRIBUTION OF THE DATASET

Category	#Instances	Category	#Instances
motorcyclist	13,485	traffic-light-green	441
car	11,696	traffic-cone	381
motorcycle	2,857	bus	369
traffic-sign	1,234	gerobak	249
pedestrian	1,092	minibus	239
traffic-light-red	796	tukang-becak	209
road-barrier	792	truck	167
pickup-truck	593	delman	109
becak	505	bicyclist	102

TABLE II. BACKBONES AND INPUT SIZES

Method	Backbone	Input size	
		Minimum	Maximum
Faster R-CNN	ResNet50	320	640
		800	1333
	MobileNetV3-Large	320	640
		800	1333
SSD	VGG16	640	640
		800	800
SSD-Lite	MobileNetV3-Large	320	320
		800	800
RetinaNet	ResNet50	800	1333
FCOS	ResNet50	640	640
		800	1333
DETR	ResNet50	480	1333
Conditional DETR	ResNet50	480	1333

Note: For DETR and Conditional DETR, the shortest side ranging from 480 to 800 pixels is simply referred to as 480 pixels.

C. Model Evaluation

The models were evaluated using the test set. This study applied the standard COCO evaluation metrics for object detection performance, consisting of mean Average Precision (mAP) at varied Intersection-over-Union (IoU) thresholds (0.5:0.95, 0.5, and 0.75) and object sizes (S: Small, M: Medium, L: Large) [15]. Additionally, computational performance was assessed on a Raspberry Pi 4 with 4 GB of RAM. The computational metrics cover inference time, Frames Per Second (FPS), CPU usage, and Resident Set Size (RSS). For computational evaluation, the trained models were exported to ONNX format and subsequently deployed on a Raspberry Pi for measurement.

III. RESULTS AND DISCUSSION

This section presents experimental results and provides a discussion, emphasizing comparative analysis of methods based on the speed-precision trade-off.

A. Detection Results

Table III illustrates the performance results of the object detection methods. Faster R-CNN with ResNet50 backbone achieved the best results with the highest mAP, mAP0.5, and mAP0.75 of 37.6%, 58.6%, and 42.2%, respectively. It was followed by FCOS, which gained the second-best performance, outperforming all the one-stage methods, with mAP, mAP0.5 and mAP0.75 of 34.8%, 50.7%, and 37.4%, respectively. Conditional DETR had the third-best results with mAP of 32.7%, followed by RetinaNet and DETR, with APs of each are 29.4% and 22.7%, respectively.

Among these methods, SSD and SSD-Lite had the lowest detection performance values. The results for SSD do not show a significant difference between the input size of 640 and that of 800, as the mAP achieved for both are 11.6% and 11.1%, respectively. Similarly, the performance of SSD-Lite with the input size of 320 does not differ significantly from that with the input size of 800, as indicated by mAP scores of 6.3% and 6.8%, respectively.

Faster R-CNN outperforms all methods on small object detection with a mAP of 21.1% (mAPS), indicating a superior performance as a two-stage detector. The performance gap is significant from FCOS, with a difference of 8.7%. The rest of the methods did not even achieve at least 10% of mAPS, especially when the input size was below 800. This also suggests that a larger input size has a significant effect on the performance of small object detection. Figure 2 shows a visualization of ground truth and predicted bounding boxes. As shown, Faster R-CNN and FCOS successfully detect objects, including small-scale instances such as traffic-sign and traffic-light-red, whereas other methods failed to detect several objects, particularly smaller ones.

B. Computational Performance on an Edge Device

Table IV displays the computational performance on the edge device (Raspberry Pi), including inference time (in seconds), FPS, CPU usage (in percentage), and RSS (in MB). As displayed, the smallest inference time was achieved by Faster R-CNN MobileNetV3 with the input size of 320, followed by SSD-Lite with the input sizes of 320 and 800, with

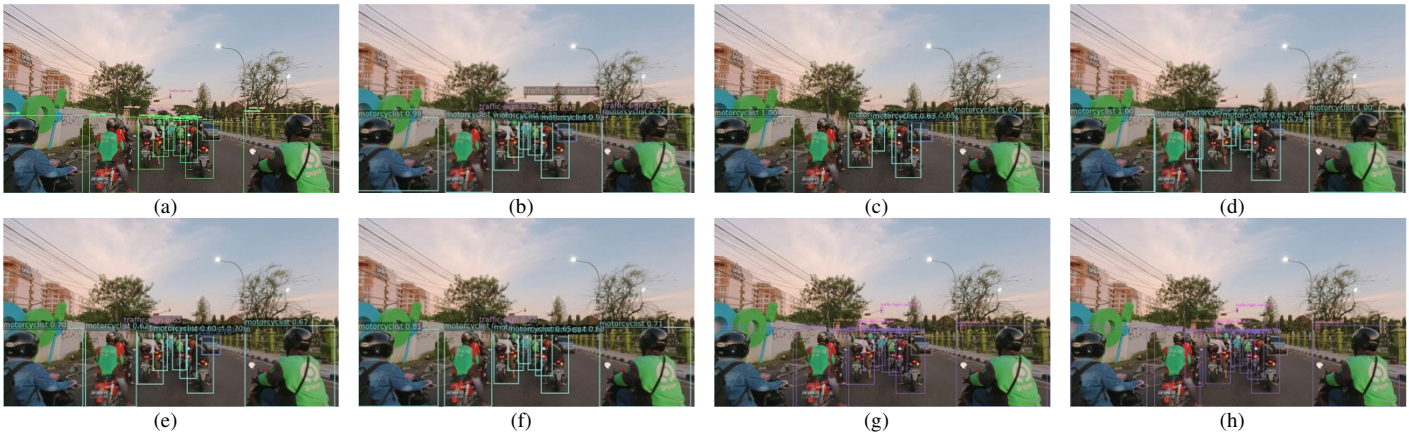
inference times of 0.58, 0.67, and 0.84 s, respectively, which are below one second. This indicates that the models can conduct inference for at least one image per second. On the contrary, Faster R-CNN ResNet50 takes a longer time for inference, i.e., 36.12 and 78.89 s for the input sizes of 320 and 800, respectively. SSD-Lite is the most lightweight model, with CPU usage and RSS values remaining below 171% and 1 GB, respectively.

C. Discussion

As a two-stage detector, Faster R-CNN has a superior performance, as indicated by the highest mAP of 37.6% and mAPS of 21.1%. The two-stage detectors generally separate detection into region proposal and refinement stages. The first stage filters out most background regions by generating object proposals, as done by the Region Proposal Network (RPN) in Faster-RCNN, allowing the second stage to focus on classifying and refining only the most promising candidates. This processing improves localization accuracy, reduces false positives, and alleviates foreground-background class imbalance, leading to higher overall precision than the one-stage detectors [2], albeit with higher computational cost.

TABLE III. DETECTION PERFORMANCE OF DIFFERENT METHODS

Method	Backbone	Input size (min)	#Parameters (millions)	mAP	mAP_{50}	mAP_{75}	mAP_S	mAP_M	mAP_L
Faster RCNN	ResNet50	320	43	**28.0**	**47.0**	**29.5**	**7.1**	**23.6**	**40.9**
		800	43	**37.6**	**58.6**	**42.2**	**21.1**	**35.3**	**48.4**
	MobileNetV3-Large	320	19	11.2	19.5	11.9	0.0	1.7	24.3
		800		21.7	38.2	23.3	1.3	12.7	39.1
SSD	VGG16	640	26	11.6	22.5	10.3	0.9	5.4	21.8
		800		11.1	21.6	9.9	1.0	5.1	21.3
SSD-Lite	MobileNetV3-Large	320	34	6.3	10.9	7.2	0.0	0.2	13.9
		800		6.8	11.0	7.3	0.0	0.1	13.4
RetinaNet	ResNet50	800	37	29.4	45.1	30.6	7.8	29.0	42.8
FCOS	ResNet50	640	32	20.8	31.8	21.7	1.4	15.1	38.8
		800		34.8	50.7	37.4	12.4	31.1	48.1
DETR	ResNet50	480	41	22.7	42.6	21.6	3.3	13.7	38.5
Conditional-DETR	ResNet50	480	43	32.7	56.6	31.8	6.4	23.8	50.5



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 2. Visualization of (a) ground truth bounding boxes, as well as detection results of (b) Faster R-CNN Resnet50 input size 800, (c) SSD VGG16 input size 640, (d) SSD-Lite MobileNetV3-large input size 320, (e) RetinaNet ResNet50 input size 800, (f) FCOS ResNet50 input size 800, (g) DETR ResNet50 input size 480, and (h) Conditional DETR ResNet50 input size 480.

TABLE IV. COMPUTATIONAL PERFORMANCE OF DIFFERENT METHODS

Method	Backbone	Input size (min)	Time (s)	CPU (%)	RSS (MB)	ONNX file size (MB)
FRCNN	ResNet50	320	36.12	372.62	1337.41	173.5
		800	78.89	367.30	1649.30	173.5
	MobileNetV3-Large	320	0.58	351.82	1016.58	76.4
		800	3.51	321.33	1350.64	76.4
SSD	VGG16	640	5.11	378.59	1080.94	104.3
		800	2.88	384.10	1039.64	104.3
SSD-Lite	MobileNetV3-Large	320	0.67	170.15	908.97	14.1
		800	0.84	161.89	909.00	14.1
RetinaNet	ResNet50	800	41.05	363.64	1391.62	147.1
FCOS	ResNet50	640	14.26	374.26	1082.37	129.0
		800	38.92	372.80	1332.42	129.0
DETR	ResNet50	480	14.77	372.38	1138.02	168.5
Conditional-DETR	ResNet50	480	17.31	369.30	1278.83	176.8

Meanwhile, one-stage detectors perform object localization and classification in a single step over dense feature maps, which makes them significantly faster. However, this dense prediction strategy must handle severe foreground-background imbalance and less selective candidate filtering, which often leads to lower precision than two-stage detectors, especially in challenging scenarios such as small or occluded objects [3, 4, 16, 17]. In addition, end-to-end Transformer-based detectors like DETR eliminate hand-crafted components such as anchor design and non-maximum suppression. While this simplifies the detection pipeline, it often comes at the cost of degraded performance on small objects [5].

Figure 3 illustrates mAP versus inference time for all the methods. Conditional DETR has a considerable precision-speed trade-off compared to the other methods. Its inference time is moderate (17.31 s), while achieving mAP comparable to FCOS-R50-800 (32.7% vs. 34.8%, respectively), whose inference time is nearly twice as long (38.92 s). This suggests that Conditional DETR is considerable for edge deployment, as it meets the trade-off between precision and computational speed. However, its performance on small objects is relatively low with mAPS of 6.4%. This is a common thing in DETR-like methods that needs improvement. More precise methods like FCOS can be another alternative for real-time detection since it has a higher mAP than Conditional DETR (34.8% vs. 32.7%, respectively) and an inference time half that of Faster R-CNN ResNet50 (38.92 vs. 78.89 s, respectively). However, for real-time application of models with higher performance and computational demands like Faster R-CNN, it is recommended to use devices with higher computational capability, particularly those equipped with a GPU, such as Jetson Xavier or Jetson Nano.

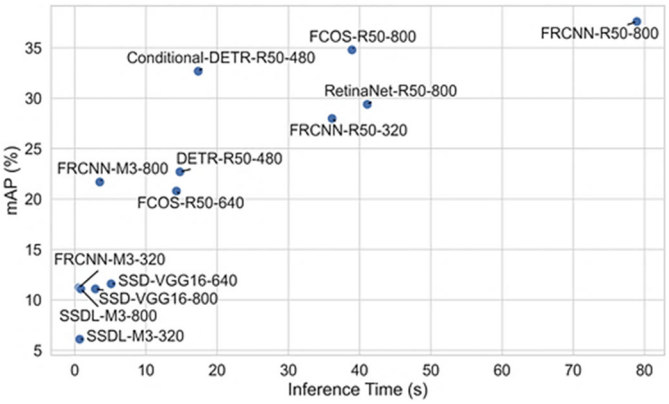


Fig. 3. Mean average precision (%) vs. inference time (s) of all methods.

IV. CONCLUSION

This study focused on the implementation of deep learning-based object detection on a resource-constrained edge device, with driving scenarios as a case study. A custom dataset was used, which contained 3,144 images of driving scenarios in Yogyakarta, Indonesia. The experimental results show that the two-stage detector Faster R-CNN with ResNet50 and the input size of 800 achieved the best mAP (37.6%) as well as mAP on small objects (mAPS: 21.1%). However, its inference time reaches 78.89 s, which is the longest among the methods. On the contrary, Faster R-CNN with MobileNetV3-Large and an input size of 320 gained the fastest inference time (0.58 s), but with poor detection performance, with mAP, mAP0.5, and mAP0.75 scores at 11.2%, 19.5%, 11.9%, respectively. Among the methods, Conditional DETR achieved a moderate inference time (17.31 s) and considerable mAP (32.7%). This suggests that Conditional DETR can be considered a more viable solution for edge deployment, as it meets the trade-off between precision and computational speed. However, its performance on small objects is still considerably low, with mAPS of 6.4%. FCOS with an input size of 800 is also an alternative due to its higher precision than Conditional DETR (34.8% vs. 32.7%) and an inference time half that of Faster R-CNN ResNet50 (twice as fast, 38.92 vs. 78.89 s). However, devices with higher computational abilities are required to better implement the deep learning-based object detection methods with higher detection performance in real-world applications.

DECLARATION OF COMPETING INTERESTS

The authors declare no competing interests that could have influenced the results of this study.

ACKNOWLEDGMENT

This work is part of a doctoral dissertation. The doctoral study is funded by the Indonesia Endowment Fund for Education (Lembaga Pengelola Dana Pendidikan, LPDP), Ministry of Finance, Republic of Indonesia.

DATA AVAILABILITY

The dataset used in this study is private but can be available by the corresponding author upon a reasonable request.

REFERENCES

[1] D. P. F. Möller and R. E. Haas, "Advanced Driver Assistance Systems and Autonomous Driving," in *Guide to Automotive Connectivity and Cybersecurity*, Springer International Publishing, 2019, pp. 513–580.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017, https://doi.org/10.1109/TPAMI.2016.2577031.

[3] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, vol. 9905, B. Leibe, J. Matas, N. Sebe, and M. Welling, Springer International Publishing, 2016, pp. 21–37.

[4] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, Feb. 2020, https://doi.org/10.1109/TPAMI.2018.2858826.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," in *Computer Vision – ECCV 2020*, vol. 12346, A. Vedaldi, H. Bischof, T. Brox, and J. M. Frahm, Springer International Publishing, 2020, pp. 213–229.

[6] D. Meng *et al.*, "Conditional DETR for Fast Training Convergence." arXiv, 2021, https://doi.org/10.48550/ARXIV.2108.06152.

[7] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable Transformers for End-to-End Object Detection," *ICLR 2021 - 9th International Conference on Learning Representations,* Austria, arXiv, 2020, https://doi.org/10.48550/ARXIV.2010.04159.

[8] G. Zhang, Z. Luo, Y. Yu, K. Cui, and S. Lu, "Accelerating DETR Convergence via Semantic-Aligned Matching," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New

Orleans, LA, USA, June 2022, pp. 939–948, https://doi.org/10.1109/CVPR52688.2022.00102.

[9] P. Kakchingtabam, B. K. Sakshi, P. Jadav, and H. Harshavardhan, "A Comprehensive Survey of Transfer Learning Techniques and Applications Across Domains," in *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*, Salem, India, Aug. 2025, pp. 1186–1195, https://doi.org/10.1109/ICSCSA66339.2025.11170934.

[10] M. Wazid, A. K. Das, V. Chamola, and Y. Park, "Uniting cyber security and machine learning: Advantages, challenges and future research," *ICT Express*, vol. 8, no. 3, pp. 313–321, Sept. 2022, https://doi.org/10.1016/j.icte.2022.04.007.

[11] S. A. El-Ghany, M. Elmogy, and A. A. A. El-Aziz, "A fully automatic fine tuned deep learning model for knee osteoarthritis detection and progression analysis," *Egyptian Informatics Journal*, vol. 24, no. 2, pp. 229–240, July 2023, https://doi.org/10.1016/j.eij.2023.03.005.

[12] C. Amisse, M. E. Jijón-Palma, and J. A. S. Centeno, "Fine-Tuning Deep Learning Models For Pedestrians Detection," *Boletim de Ciências Geodésicas*, vol. 27, no. 2, 2021, Art. no. e2021013, https://doi.org/10.1590/s1982-21702021000200013.

[13] H. Rahmat, S. Wahjuni, and H. Rahmawan, "Performance Analysis of Deep Learning-based Object Detectors on Raspberry Pi for Detecting Melon Leaf Abnormality," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 12, no. 2, pp. 572–579, Apr. 2022, https://doi.org/10.18517/ijaseit.12.2.13801.

[14] H. Rahmat, Suprapto, and M. E. Wibowo, "Concatenated feature pyramid network for robust object detection in urban driving scenarios: A case study in Yogyakarta, Indonesia," *Applied Soft Computing*, vol. 196, June 2026, Art. no. 115098, https://doi.org/10.1016/j.asoc.2026.115098.

[15] T. Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, vol. 8693, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Springer International Publishing, 2014, pp. 740–755.

[16] B. Karbouj, G. A. Topalian-Rivas, and J. Krüger, "Comparative Performance Evaluation of One-Stage and Two-Stage Object Detectors for Screw Head Detection and Classification in Disassembly Processes," *Procedia CIRP*, vol. 122, pp. 527–532, 2024, https://doi.org/10.1016/j.procir.2024.01.077.

[17] H. Bi, V. Wen, and Z. Xu, "Comparing one-stage and two-stage learning strategy in object detection," *Applied and Computational Engineering*, vol. 5, no. 1, pp. 171–177, May 2023, https://doi.org/10.54254/2755-2721/5/20230556.