COMPARISON OF YOLOV8 AND PYTORCH-RETINANET FOR VEHICLE DETECTION

Bui Xuan Tung[1*], Trinh Quang Minh[1], Ngo Thi Lan[1], Dang Thi Dung[2], Huynh Duy Dang[2]

[1]Tay Do University, [2]Can Tho University of Engineering - Technology

ARTICLE INFO

Received: 23/01/2025

Revised: 11/03/2025

Published: 21/03/2025

KEYWORDS

YOLOv8

PyTorch-RetinaNet

Vehicle detection

Machine learning

Deep learning

ABSTRACT

This study aims to evaluate and compare the effectiveness of two deep learning models - PyTorch-RetinaNet and YOLOv8 - for vehicle detection, addressing the challenges in object detection across varying size, shape, and lighting conditions. The research methodology utilized a comprehensive dataset of 4,058 vehicle images with 12 distinct object classes, implementing both models with varying learning rates (0.001, 0.01, and 0.0001). The dataset was split into training (65%), validation (24%), and testing (11%) sets, with preprocessing techniques including image resizing, brightness normalization, and data augmentation applied to enhance model performance. The experimental results revealed distinct capabilities for each model: PyTorch-RetinaNet achieved a mAP50 of 38.6% and mAP50-95 of 24.7%, exhibiting particular strength in detecting large objects (mAP50-95 of 42.0%) and maintaining stable recall metrics (AR@1: 30.9%, AR@10: 54.7%, AR@100: 55.9%). In contrast, YOLOv8 demonstrated superior overall performance with a mAP50 of 45.6%, mAP50-95 of 33.0%, precision of 48.3%, and recall of 61.5%, particularly excelling in handling overlapping objects with confidence scores of 0.79-0.89. The findings suggest YOLOv8 is more suitable for real-time applications, while PyTorch-RetinaNet excels in scenarios requiring precise detection across varying object sizes.

SO SÁNH YOLOV8 VÀ PYTORCH-RETINANET TRONG PHÁT HIỆN PHƯƠNG TIỆN GIAO THÔNG

Bùi Xuân Tùng[1*], Trịnh Quang Minh[1], Ngô Thị Lan[1], Đặng Thị Dung[2], Huỳnh Duy Đặng[2]

[1]Trường Đại học Tây Đô, [2]Trường Đại học Kỹ thuật – Công nghệ Cần Thơ

THÔNG TIN BÀI BÁO

Ngày nhận bài: 23/01/2025

Ngày hoàn thiện: 11/03/2025

Ngày đăng: 21/03/2025

TỪ KHÓA

YOLOv8

PyTorch-RetinaNet

Phát hiện phương tiện giao thông

Học máy

Học sâu

TÓM TẮT

Nghiên cứu này nhằm mục đích đánh giá và so sánh hiệu quả của hai mô hình học sâu - PyTorch-RetinaNet và YOLOv8 để phát hiện phương tiện, giải quyết các thách thức trong việc phát hiện đối tượng trên nhiều kích thước, hình dạng và điều kiện ánh sáng khác nhau. Phương pháp nghiên cứu sử dụng một tập dữ liệu gồm 4.058 hình ảnh xe với 12 lớp đối tượng riêng biệt, triển khai cả hai mô hình với tốc độ học khác nhau (0,001, 0,01 và 0,0001). Tập dữ liệu được chia thành các tập huấn luyện (65%), xác thực (24%) và thử nghiệm (11%), với các kỹ thuật xử lý trước bao gồm thay đổi kích thước hình ảnh, chuẩn hóa độ sáng và tăng cường dữ liệu được áp dụng để nâng cao hiệu suất của mô hình. Kết quả thử nghiệm cho thấy mô hình PyTorch-RetinaNet đạt được mAP50 là 38,6% và mAP50-95 là 24,7%, đặc biệt trong việc phát hiện các vật thể lớn (mAP50-95 là 42,0%) và duy trì số liệu thu hồi ổn định (AR@1: 30,9%, AR@10: 54,7%, AR@100: 55,9%). Ngược lại, YOLOv8 cho thấy hiệu suất tổng thể vượt trội với mAP50 là 45,6%, mAP50-95 là 33,0%, độ chính xác là 48,3% và khả năng thu hồi là 61,5%, đặc biệt trong việc xử lý các đối tượng chồng chéo với điểm tin cậy là 0,79-0,89. Các phát hiện cho thấy YOLOv8 phù hợp hơn với các ứng dụng thời gian thực, trong khi PyTorch-RetinaNet nổi bật hơn trong các tình huống đòi hỏi phát hiện chính xác trên các kích thước đối tượng khác nhau.

DOI: https://doi.org/10.34238/tnu-jst.11942

[*] Corresponding author. *Email: bxtung@tdu.edu.vn*

1. Introduction

Object detection and classification face significant challenges in handling diverse object sizes, shapes, and lighting conditions, particularly in vehicle detection applications. These challenges are compounded by the limited availability of comprehensive datasets representing real-world scenarios. While convolutional neural networks (CNNs) have shown promising results in object detection [1], achieving high accuracy and processing speed remains a critical challenge. PyTorch-RetinaNet with ResNet-50 [2], [3] and YOLOv8 with Darknet-53 [4] - perform specifically in vehicle detection scenarios where objects frequently overlap and vary in size. This study aims to address this gap by systematically comparing both models using a diverse vehicle dataset of 4,058 images [5], evaluating their performance across different operating conditions to provide clear insights for selecting appropriate architectures in vehicle detection applications.

Recent research has explored various solutions to these challenges. Studies comparing RetinaNet and YOLOv3 for real-time detection tasks have shown that RetinaNet achieves higher accuracy (mAP: 82.89%) but slower processing (17 FPS), while YOLOv3 offers faster detection (51 FPS) with competitive accuracy (mAP: 80.69%) [6]. Additional research by Nife et al [7] has confirmed these trade-offs between speed and accuracy in various application scenarios. Reis [8] et al presents two YOLOv8 models for real-time flying object detection: a generalized model trained on 40 object classes (mAP50: 79.2%, mAP50-95: 68.5%, 50 FPS at 1080p) and a refined model using transfer learning for real-world conditions (mAP50: 99.1%, mAP50-95: 83.5%, 50 FPS at 1080p). Recent studies advance object detection: Tan et al [9] evaluates YOLOv3 and SSD for unmanned driving, Guo et al [10] improves YOLOv8n-seg with FasterNet, CBAM, and WIoU loss function achieving 98.3% car detection accuracy. At the same time, YOLOv8's architecture demonstrates effectiveness in detecting flying objects with varying conditions.

2. Proposed method

2.1. Problem Model

2.1.1. Yolo

Figure 1 illustrates the process of using the YOLOv8 model to detect vehicles on the road. The upper part shows a frame from a traffic surveillance camera, reflecting a street scene with multiple moving vehicles. YOLOv8 analyzes the frame to identify the vehicles.

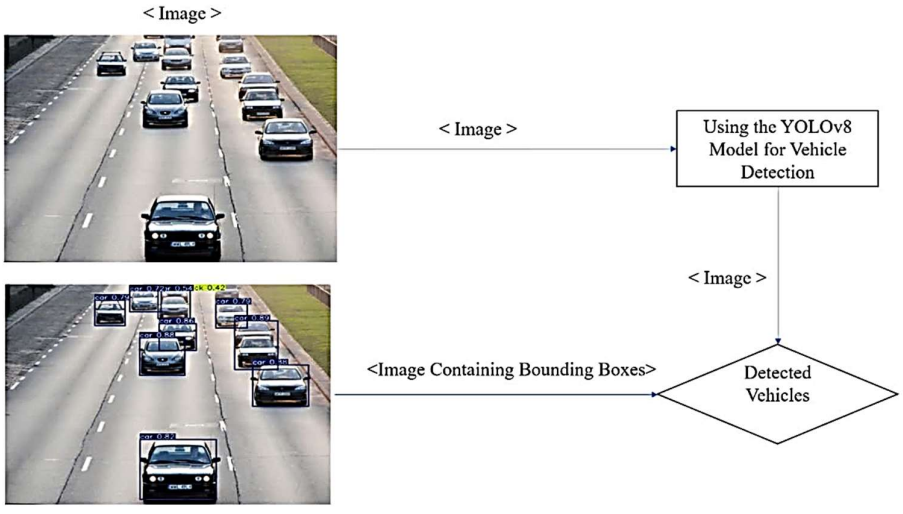

Figure 1. *YOLO Model*

The model processes the input image and determines vehicle locations. The result is shown in the lower part, where the original frame is updated with bounding boxes around detected vehicles. These boxes help users easily identify and track vehicles on the road, demonstrating the effectiveness of object detection technology in traffic monitoring.

2.1.2. RetinaNet

RetinaNet is a deep neural network architecture that integrates ResNet and Feature Pyramid Networks (FPN) for object detection (Figure 2). The processing begins when the input image (such as a snowy landscape with objects like cars and trees) is passed through ResNet - a powerful CNN that extracts key features from raw pixels into elements like edges, corners, and shapes [11]. Next, FPN constructs features at multiple levels, allowing the model to process objects of various sizes, from large (like vehicles) to small (like license plates) [12]. These features are then fed into two specialized sub- networks: a classification network for identifying object labels and a regression network for predicting precise locations through bounding box generation. RetinaNet employs the Focal Loss function to handle the imbalance between background and object samples during training while applying Non-Maximum Suppression (NMS) to eliminate duplicate boxes, retaining only the highest-confidence box. The result is an output image with objects marked by bounding boxes and classification labels, demonstrating RetinaNet's superior performance in complex object detection tasks.

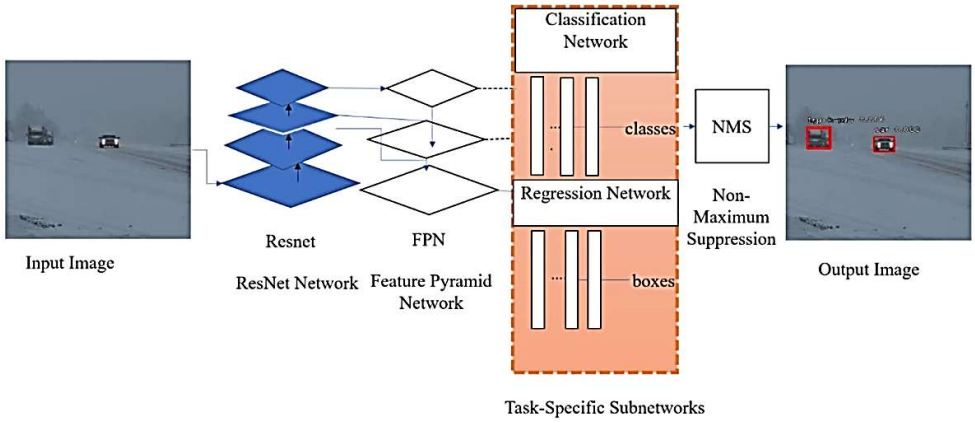

Figure 2. *RetinaNet Model*

2.2. Dataset configuration

This study uses the vehicles dataset [13], comprising 4,058 images with 12 detailed object annotations, including various vehicle types such as cars, trucks, containers, pickups, and buses. Due to memory limitations, the data was optimized to ensure efficient storage and processing. Dataset details, including the number of images per object class and the distribution across training, validation, and testing sets, are presented in Table 1.

Table 1. *Dataset structure*

Data Classification	Number of Images	Percentage (%)
Train	2634	65%
Valid	966	24%
Test	458	11%

Annotations for each image are provided in XML or JSON files, while images are stored in JPG format. Each annotation includes information about bounding box coordinates, object class (label), and the confidence level of each object. Bounding box coordinates are defined based on the image's coordinate system, enabling precise object localization.

2.3. Experimental Platform

The experiments were conducted on Google Colab [14] with the following hardware specifications: CPU: Intel Xeon E5-2667 v3; GPU: Nvidia P40-24Q with 24 GB GDDR5 PCIe 3.0 memory; RAM: 48 GB; Storage: 300 GB SSD.

3. Results and Discussion

3.1. Results after training the Model

3.1.1. Performance of the YOLOv8 Model

We used YOLO integrated with Ultralytics as an encoder to optimize the training process to extract features from input images. This approach eliminates the need for the model to learn basic features from scratch, reducing the complexity of the training process and the number of parameters to optimize. During experimentation, we used different learning rate (lr) values, including 0.001, 0.005, and 0.01, along with a decay mechanism to determine the optimal rate for the working environment. If the learning rate is too high, the model risks skipping the global optimum; conversely, if the rate is too low, training can become slow or get stuck at a local optimum.

The training results of the YOLO model on the "Vehicle" dataset Figure 3 showed significant improvements across epochs. Train/box_loss decreased from 1.4 to 1.0, train/cls_loss dropped from 1.75 to 0.5, and train/dfl_loss reduced from 1.15 to 1.0 after 30 epochs, indicating the model gradually learned to identify bounding boxes and classify labels more accurately. On the validation set, val/box_loss decreased from 1.4 to 1.25, and val/cls_loss fell from 1.3 to 1.1, despite slight fluctuations in the early stages. Precision and recall reached 0.6 and 0.65 by the end of training, reflecting good accuracy and object detection capability.

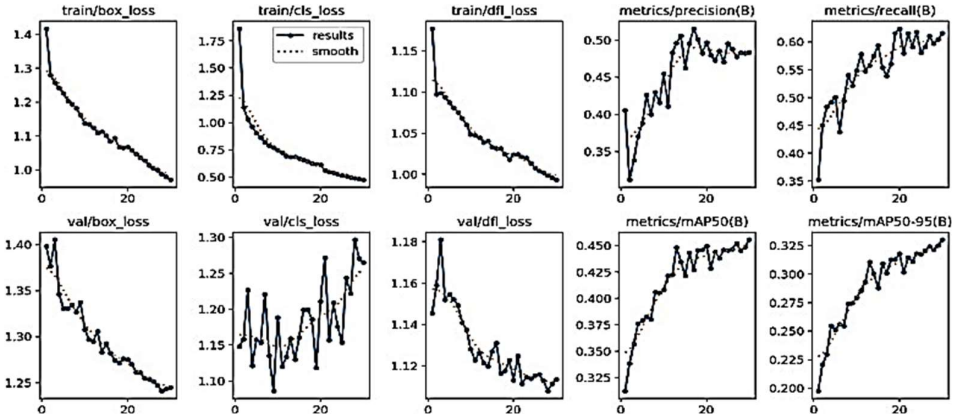

Figure 3. *Results obtained from the Yolov8 Model*

Table 2. *Performance of the Yolov8 Model*

Metric	Epoch 1	Epoch 15	Epoch 30
GPU_mem	4.22G	4.09G	3.96G
Box_loss	1.418	1.101	0.9708
Cls_loss	1.861	0.6725	0.4767
Dfl_loss	1.177	1.034	0.9935
Instances	160	122	128
Size	640	640	640
Box(P)	0.405	0.463	0.484
Box(R)	0.353	0.594	0.615
mAP50	0.312	0.422	0.456
mAP50-95	0.198	0.288	0.33

The mAP50 steadily increased from 0.2 to 0.45, while mAP50-95 rose from 0.2 to 0.325, indicating that the model's performance across various IoU thresholds still requires improvement. These results confirm that the model performs effectively but requires further optimization to enhance critical metrics like mAP50-95 (Table 2).

3.1.2. Performance of the PyTorch-RetinaNet Model

To optimize the training process, we combined Pytorch-RetinaNet with ResNet50 to use as a feature extraction encoder for the input images. That way, the model does not need to learn the base features from scratch, reducing the complexity of the training process and the total number of parameters that need to be trained. We experimented with different learning rates (lr), including 0.001, 0.005, and 0.01 and decreasing, to choose the best rate for the environment. Too high a learning rate can cause the model to "jump" over the optimal point, while too low a learning rate can lead to slow learning or getting stuck in a local optimum.

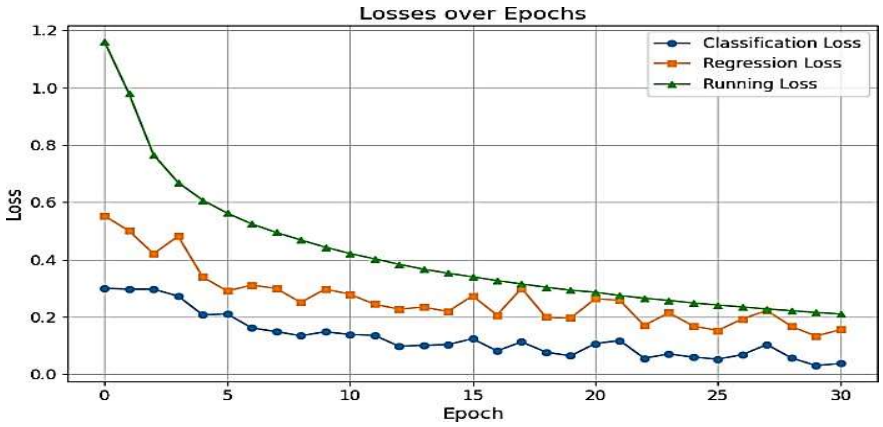

Figure 4. *Line chart showing loss over epochs*

Figure 4 shows a clear downward trend in loss values throughout the training process, reflecting the model's improved efficiency over time. Specifically, the Classification Loss started at approximately 0.25 and steadily decreased, falling below 0.2 by the end of training. This indicates that the model has increasingly accurately classified objects in images. The training process of the model over 30 epochs (Table 3) shows a significant performance improvement. In the early stage (Epoch 0), Running Loss starts at a peak of about 1.15, while Classification Loss and Regression Loss are 0.29986 and 0.55224, respectively.

Table 3. *Performance of the RetinaNet Model*

Metric	Epoch 0	Epoch 15	Epoch 30
Iteration	219	219	219
Classication Losss	0.29986	0.12349	0.03676
Regression Loss	0.55224	0.27319	0.15524
Running Loss	1.16069	0.33964	0.21001

This shows that the initial model still has many prediction errors. Notably, the fastest loss reduction rate occurs in the first 5 epochs, especially the Running Loss drops sharply from 1.16069 to about 1.16069. This proves that the model learns very effectively in the early training stage. After that, the loss reduction rate slows down and becomes more stable from epoch 15 onwards. At epoch 15, the loss indices have decreased significantly with Running Loss at 0.33964, Classification Loss at 0.12349 and Regression Loss at 0.27319. At the last epoch (epoch 30), the model achieved the best performance with Running Loss at 0.21001, Classification Loss at

0.03676 and Regression Loss at 0.15524. In particular, the number of iterations remained unchanged (remained at 219) across epochs, indicating that the dataset size and batch size were kept stable throughout the training process. The decreasing and stable trend of the loss curves also indicates that the model has converged well and there is no sign of overfitting.

3.2. Comparison of PyTorch-RetinaNet and YOLOv8

Case with a Single Vehicle in the Image: In this scenario, both YOLO and RetinaNet successfully detected the vehicle but RetinaNet (Figure 5) showed higher reliability with a confidence score of 0.91 compared to YOLO 0.65 (Figure 6).

Figure 5. *Detection of a Single Object by RetinaNet*

Figure 6. *Detection of a Single Object by YOLO*

Case with Multiple Overlapping Vehicles:

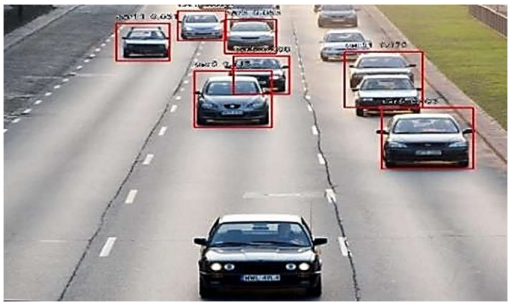

Figure 7. *Detection of Multiple Objects with RetinaNet*

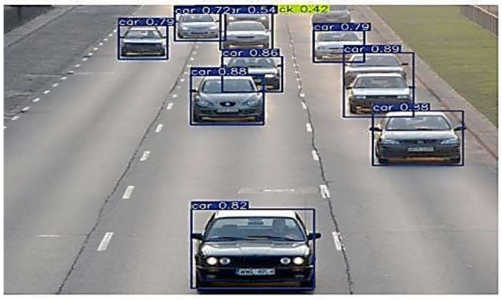

Figure 8. *Detection of Multiple Objects with YOLO*

In situations with multiple overlapping or occluded vehicles, YOLO demonstrated superior performance compared to RetinaNet. YOLO (Figure 8) detected 9 out of 12 objects with high confidence (0.79 to 0.89), whereas RetinaNet (Figure 7) detected only 7 out of 12 objects with lower confidence (0.51 to 0.002).

Case with Small or Blurry Vehicles:

Figure 9. *Detection of Small or Blurry Objects with RetinaNet*

Figure 10. *Detection of Small or Blurry Objects with Yolo*

RetinaNet generally outperformed YOLO in detecting small or blurry objects. Although YOLO had higher confidence scores (0.45 compared to RetinaNet's 0.056 and 0.083), it misclassified the objects, while RetinaNet (Figure 10) correctly identified both objects but with lower confidence.

3.3. Overview of Comparison

By analyzing the performance of the two object recognition models YOLOv8 and Pytorch-RetinaNet shown in Table 4, we can see that each model shows its advantages in different situations. For explicit objects, RetinaNet shows superior performance with a score of 0.91, significantly higher than YOLOv8 (0.65), although both detect 1/1 objects correctly. This difference stems from the ResNet-50 architecture that combines FPN and Focal Loss of RetinaNet, allowing for multi-scale feature fusion and focusing on difficult patterns. However, when dealing with overlapping and occluded objects, YOLOv8 performs much better with scores ranging from 0.79 to 0.89 and detecting 9/12 objects, while RetinaNet only scores from 0.088 to 0.002 and detecting 7/12 objects. This advantage of YOLOv8 comes from the Darknet-53 architecture with CSP blocks, a single-stage prediction method, and mosaic augmentation techniques during training. For the case of small and blurry objects, although YOLOv8 has a higher score (0.45 compared to 0.056-0.083 of RetinaNet), RetinaNet detects more objects (2/2 compared to 1/2) thanks to the diverse anchor box mechanism and FPN. This shows that a high score does not always mean better detection, and also makes it clear why YOLOv8 is particularly suitable for real-time applications with overlapping objects, while RetinaNet performs better with clear and small objects. The choice of which model will depend on the specific requirements of the application and the characteristics of the objects to be detected.

Table 4. *Comparison of Yolov8 and Pytorch-RetinaNet*

Criteria	YOLOv8	PyTorch- RetinaNet	Number of Objects Detected
Clear objects	0.65	0.91	1/1 (YOLO and RetinaNet)
Overlapping objects	0.79 to 0.89	0.088 to 0.002	9/12 (YOLO), 7/12 (RetinaNet)
Small, blurry objects	0.45	0.056 to 0.083	1/2 (YOLO), 2/2 (RetinaNet)

4. Conclusion

Processing image data poses a significant challenge due to its diversity and complexity, including variations in features, ambiguities in label definitions, and the influence of external factors such as lighting and background. In this study, the performance of two models, PyTorch-RetinaNet and YOLOv8, is compared for object detection.

PyTorch-RetinaNet achieved a mAP50-95 of 24.7%, a mAP50 of 38.6%, and a mAP75 of 27.9%. It performed better with larger objects, achieving a mAP50-95 of 42%. In terms of recall, the model reached AR@1 of 30.9%, AR@10 of 54.7%, and AR@100 of 55.9%.

Meanwhile, YOLOv8 achieved a mAP50-95 of 33%, a mAP50 of 45.6%, a precision of 48.3%, and a recall of 61.5%. Although YOLOv8 had a lower mAP50-95 for larger objects, it stood out in precision and recall, enabling effective detection even in complex scenarios such as overlapping or occluded objects.

This study makes significant new contributions compared to previous works. While previous studies such as Tan et al. [6] or Nife and Chtourou [7] focused on the old YOLOv3 and RetinaNet versions, our study is one of the first to evaluate in detail the performance of YOLOv8 - the latest version with many improvements in network architecture and algorithms - against PyTorch-RetinaNet. In particular, we not only compare general performance metrics but also analyze them in detail in three specific real-world scenarios (clear, overlapping/occluded, and small/blurry objects), combining both quantitative analysis and visual illustrations. With a focus on vehicle detection of 12 different types, the study provides a deeper understanding of the

strengths and weaknesses of each model in smart traffic monitoring applications [10], going beyond the general approach commonly found in previous studies [13]. Our results not only contribute to the theory of object detection but also provide practical guidance for selecting appropriate models in traffic monitoring applications based on the specific characteristics of each situation [11], [12].

REFERENCES

[1] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, 2019.

[2] G. Tan, Z. Guo, and Y. Xiao, "PA-RetinaNet: Path augmented RetinaNet for dense object detection," in *International Conference on Artificial Neural Networks*, 2019, pp. 138-149.

[3] B. Koonce and B. Koonce, "ResNet 50," in *Convolutional Neural Networks with Swift for TensorFlow: Image Recognition and Dataset Categorization*, 2021, pp. 63-72.

[4] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with YOLOv8," *arXiv preprint arXiv:2305.09972*, pp. 1-12, 2023.

[5] S. Alexandrova, Z. Tatlock, and M. Cakmak, "RoboFlow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5537-5544.

[6] L. Tan, T. Huangfu, L. Wu, and W. Chen, "Comparison of RetinaNet, SSD, and YOLO v3 for Real-Time Pill Identification," *BMC Medical Informatics and Decision Making*, vol. 21, pp. 1- 11, 2021.

[7] N. I. Nife and M. Chtourou, "A Comprehensive Study of Deep Learning and Performance Comparison of Deep Neural Network Models (YOLO, RetinaNet)," *International Journal of Online & Biomedical Engineering*, vol. 19, no. 12, pp. 456-469, 2023.

[8] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with YOLOv8," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, 2023.

[9] L. Tan, T. Huangfu, L. Wu, and W. Chen, "Comparison of RetinaNet, SSD, and YOLO v3 for Real-Time Pill Identification," *IEEE Transactions on Medical Imaging*, vol. 21, no. 1, pp. 1- 11, 2021.

[10] H. Guo, Y. Zhang, L. Chen, and A. A. Khan, "Research on vehicle detection based on improved YOLOv8 network," *arXiv preprint arXiv:2501.00300*, pp. 1-8, 2024.

[11] Y. Li, S. Zhou, and H. Chen, "Attention-based fusion factor in FPN for object detection," *Applied Intelligence*, vol. 52, no. 13, pp. 15547-15556, 2022.

[12] N. Wulandari, I. Ardiyanto, and H. A. Nugroho, "A Comparison of Deep Learning Approach for Underwater Object Detection," *Journal of Engineering Systems and Information Technology,* vol. 6, no. 2, pp. 252-258, 2022.

[13] Z. Luo, F. Branchaud-Charron, C. Lemaire, J. Konrad, S. Li, A. Mishra, and P. M. Jodoin, "MIO-TCD: A new benchmark dataset for vehicle classification and localization," *IEEE Transactions on Image Processing*, vol. 27, no. 10, pp. 5129-5141, 2018.

[14] X. Pan, R. Snyder, J. N. Wang, C. Lander, C. Wickizer, R. Van, and Y. Shao, "Training machine learning potentials for reactive systems: A Colab tutorial on basic models," *Journal of Computational Chemistry*, vol. 45, no. 10, pp. 638-647, 2024.