DAC-204

YOLO: A Comparative Study in Object Detection Algorithms

Vidit Aggarwal, *23125037*, Pratham Patel, *23125025*

***Abstract*—This study presents a comprehensive comparative analysis of recent advancements in object detection, focusing primarily on the YOLO (You Only Look Once) models—YOLOv8 and the newly released YOLOv9 & YOLOv10—against traditional two-stage detectors like FasterRCNN and transformer-based models such as DINO (Detection with Improved Network Optimization). YOLO models, known for their real-time inference capabilities and architectural efficiency, have undergone significant improvements from version 1 to version 10. YOLOv9 introduces novel structural changes, including an enhanced backbone and improved attention mechanisms, resulting in better accuracy and generalization without compromising speed.**

Our experiments evaluate these models on a custom dataset, considering key metrics such as mean Average Precision (mAP), f1 score, and precision/recall. YOLOv9,v10 shows a marked improvement over YOLOv8 in both mAP and detection speed, affirming its superiority in real-time applications. However, when compared to two-stage detectors like Faster RCNN, YOLO models still trade off a slight amount of accuracy for faster inference.

Moreover, DINO, which leverages Vision Transformers (ViTs), demonstrates state-of-the-art detection performance, especially in complex scenes and small object detection. However, this comes at the cost of significantly higher computational requirements and latency, making it less suitable for edge or real-time deployment.

I. INTRODUCTION

THIS report contains our research on the comparative analysis of the different Object Detection algorithms, taking YOLO algorithm as our baseline.

Accurate and efficient object detection is fundamental to autonomous driving and advanced driver-assistance systems (ADAS), where timely recognition of vehicles, pedestrians, and road signs underpins safe navigation in diverse environments 20. Convolutional neural network (CNN)-based detectors have been at the forefront of this progress, offering a balance between high precision and inference speed necessary for real-time deployment 20. Yet, the choice of detector architecture often involves trade-offs: single-stage methods excel in speed, while two-stage and transformer-based approaches typically yield higher localization accuracy at the cost of latency.

You Only Look Once (YOLO) ushered in the era of high-speed, single-stage detection by framing object detection as a unified regression problem, achieving upwards of 45 FPS with

competitive mean Average Precision (mAP) on datasets such as PASCAL VOC and COCO (21,22). Subsequent YOLO iterations have continuously advanced the speed-accuracy frontier, making YOLO models a staple for applications demanding real-time performance 1.

FasterR-CNN established the two-stage detection paradigm by integrating a Region Proposal Network (RPN) that shares convolutional features with the downstream detection head, effectively rendering region proposal generation nearly cost-free and improving localization precision, particularly for small or occluded objects 1. While this approach typically incurs higher inference latency compared to YOLO, it remains a strong benchmark for applications where detection accuracy is paramount.

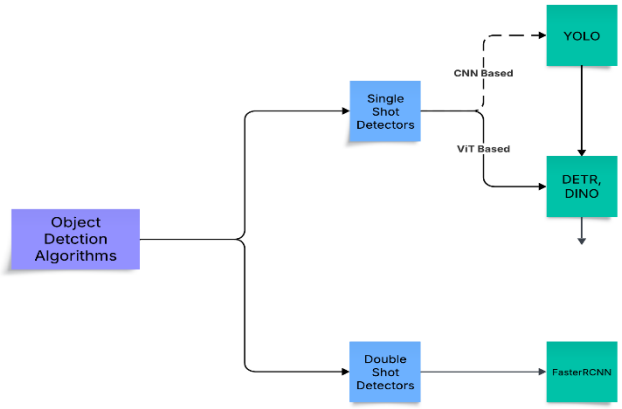


Fig. 1. Classification of Object Detection Algorithms

More recently, transformer-based detectors such as DETR have reframed object detection as a direct set prediction task, eliminating the need for hand-crafted components like NMS or anchor design. DINO (DETR with Improved DeNoising Anchor Boxes) further enhances convergence and performance through a denoising training scheme, mixed query initialization, and a “look forward twice” box prediction strategy.(4,5).

Despite the breadth of benchmarks on generic datasets such as COCO, these collections underrepresent severe weather and region-specific road characteristics 6.

To address this gap, we introduce a custom Canadian Road dataset and conduct a threefold comparative study:

(1) **intra-family analysis** of YOLOv8, YOLOv9, and YOLOv10;

(2) **cross-paradigm benchmarking** of YOLO versus Faster R-CNN; and

(3) **transformer-based evaluation** comparing YOLO to

The code for our implementations can be found at: GitHub

DAC-204

DINO.

II. The Video Presentation

In our video presentation, we began with an introduction to how object detection works using the YOLO (You Only Look Once) framework. We explained its real-world applications in areas like surveillance, autonomous vehicles, and industrial safety. We then discussed the core concept of YOLO's detection pipeline, including how it divides the image into grids, predicts bounding boxes, class probabilities, and uses Intersection over Union (IoU) to evaluate the accuracy of predicted boxes. Following this, we provided an overview of the YOLO architecture, highlighting its single-shot detection approach and the role of convolutional layers in feature extraction and object localization. We then delved into the mathematics behind YOLO's loss functions, breaking down the components for localization, confidence, and classification loss, and explaining how they guide the training process. Finally, to ground the concepts in a practical setting, we demonstrated an example of helmet detection using YOLO, showing how the model can be trained on a custom dataset to identify safety gear in real-time. This example showcased the power and versatility of YOLO in real-world applications, especially when paired with custom-labeled data.

III. The YOLO Algorithm

You Only Look Once (YOLO) formulates object detection as a single, unified regression task that directly maps an input image to bounding-box coordinates and class probabilities in one network evaluation. Given image I, such that:

$$I \in \mathbb{R}^{3\times H\times W}$$

YOLO first resizes it to a fixed resolution (e.g. 448×448) and passes it through a deep convolutional backbone-24 convolutional layers interleaved with four max-pooling layers-followed by two fully connected layers. The final output is a tensor:

$$Y = f_\theta(I) \in \mathbb{R}^{S\times S\times(B\cdot 5+C)}$$

where *S*×*S* is the grid size, *B* the number of boxes per cell, and *C* the number of classes.

Grid Representation and Bounding-Box Parametrization:

YOLO divides the feature map into an *S*×*S* grid. Each cell (*i*,*j*) predicts:

- *B* bounding boxes, each with parameters:

  $(x_{ijb},\ y_{ijb},\ w_{ijb},\ h_{ijb},\ C_{ijb})$ for $b = 1, \ldots, B$

  Where,

  $(x_{ijb},\ y_{ijb}) \in [0,1]^2$

  are the box centre offsets relative to the cell's top-left corner,

  $(w_{ijb},\ h_{ijb}) \in [0,1]^2$

  are the width and height normalized by the image dimensions,

  $C_{ijb} = \Pr(\text{object}) \times \text{IoU}(\text{pred}, \text{truth})$

  is the "confidence."

- A conditional class probability vector

  $(p_{ij1}, \ldots, p_{ijC})$

  where

  $p_{ijc} = \Pr(c \mid \text{object})$

At inference, the class-specific score for box *b* in cell (*i*,*j*) is

$$s_{ijb}(c) = C_{ijb} \times p_{ijc} = \Pr(\text{object}) \times \text{IoU} \times \Pr(c \mid \text{object})$$

and detections below a confidence threshold are discarded before applying Non-Maximum Suppression.

Loss Function:

During training, each ground-truth box:

$(x^\wedge, y^\wedge, w^\wedge, h^\wedge, c^\wedge)$

is assigned to the grid cell containing its center and to the box predictor *b* with maximal IoU. Denote:

- $1_{ijobj,b} = 1$ if cell (i, j) and box b is responsible for prediction, else 0.
- $1_{ijnoobj,b} = 1 - 1_{ijobj,b}$
- $1_{ijobj} = \sum_b 1_{ijobj,b}$
- $1_{ijnoobj} = \sum_b 1_{ijnoobj,b}$

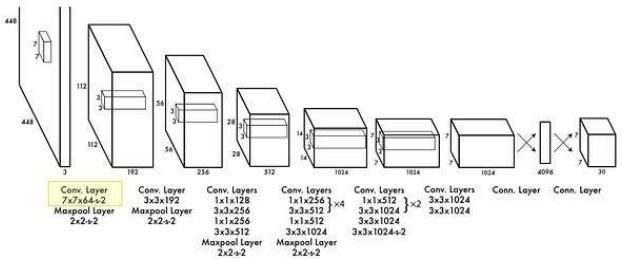
The total loss is

$$\begin{aligned} L = {} & \lambda_{\text{coord}} \sum_{i,j} \sum_b 1_{ijobj,b} \left[(x_{ijb} - \hat{x}_{ijb})^2 + (y_{ijb} - \hat{y}_{ijb})^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i,j} \sum_b 1_{ijobj,b} \left[(w_{ijb} - \hat{w}_{ijb})^2 + (h_{ijb} - \hat{h}_{ijb})^2 \right] \\ & + \sum_{i,j} \sum_b 1_{ijobj,b} (C_{ijb} - \hat{C}_{ijb})^2 \\ & + \lambda_{\text{noobj}} \sum_{i,j} \sum_b 1_{ijnoobj,b} (C_{ijb} - \hat{C}_{ijb})^2 \\ & + \sum_{i,j} 1_{ijobj} \sum_c (p_{ijc} - \hat{p}_{ijc})^2 \end{aligned}$$

Here λ_{coord} (e.g. 5) up-weights localization errors, while λ_{noobj} (e.g. 0.5) down-weights confidence errors on background boxes.

ARCHITECTURE

VISUALIZATION OF YOLO



The network processes an input image of size 448×448 and

passes it through a series of convolutional and pooling layers to extract spatial features. The architecture begins with a 7×7 convolutional layer with 64 filters followed by max pooling, which reduces the spatial dimensions while retaining important features.

Subsequent layers consist of multiple 3×3 and 1×1 convolutional layers with increasing depth (up to 1024 filters), interleaved with pooling layers to progressively downsample the feature map. The 1×1 convolutions serve as bottleneck layers that reduce dimensionality and computation.

After feature extraction, the high-dimensional feature map is flattened and passed through two fully connected layers, including a dense layer with 4096 units, and finally to an output layer that predicts 30 values per grid cell. These values correspond to bounding box coordinates, confidence scores, and class probabilities.

IV. Double Shot & ViT Based Detectors

A. Double Shot Detection & FasterRCNN

FasterR-CNN is a two-stage object detection framework that integrates a Region Proposal Network (RPN) with a FastR-CNN detection head over shared convolutional features to enable efficient, end-to-end learning and inference 3. It first generates object proposals via a fully convolutional RPN and then refines and classifies these proposals through RoI pooling and specialized detection heads 3.

i. **Backbone & Feature Extraction**
A deep convolutional network (e.g., VGG-16 or ResNet-50) processes the input image to produce a dense feature map, which serves as the backbone for both proposal generation and object classification3.

ii. **Region Proposal Network (RPN)**
The RPN slides a small *n x n* window (typically 3×3) over the backbone feature map and, at each spatial location, generates K anchor boxes with multiple scales and aspect ratios3. For each anchor, it predicts:

- An "objectness" score via a binary classification branch.
- Bounding-box regression offsets (t_x, t_y, t_w, t_h) to adjust the anchor coordinates.

Top-scoring proposals are filtered by score thresholding and Non-Maximum Suppression to yield a fixed set of region proposals for the detection head3.

iii. **RoI Pooling**
Each region proposal is projected onto the shared feature map and pooled into a fixed spatial size (e.g. 7×7) using RoI Pooling, which divides the proposal into equal bins and applies max pooling in each bin to produce a uniform feature tensor3.

iv. **Detection Heads**
RoI-pooled features pass through two parallel branches:

- A classification head with a softmax over C+1 categories.
- A bounding-box regression head that outputs class-specific coordinates refinements.

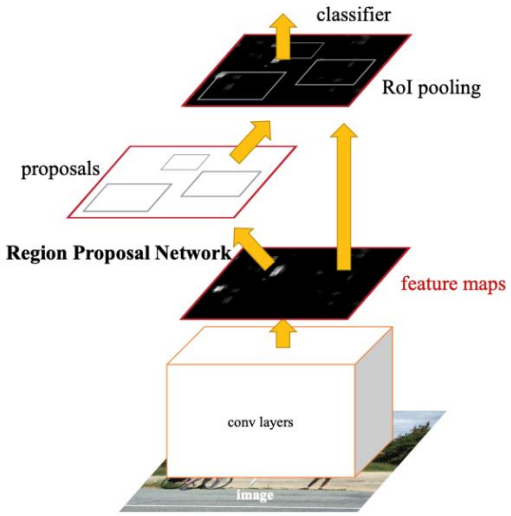


Fig. 2. The FasterRCNN Algorithm

Training Objective

Faster R-CNN employs a multi-task loss combining RPN and detection losses:

$$L = \frac{1}{N_{\text{cls}}} \sum L_{\text{cls}} + \lambda \frac{1}{N_{\text{reg}}} \sum L_{\text{reg}}$$

where L_{cls} is the cross-entropy loss for objectness or class prediction, L_{reg} is the smooth L1 loss for bounding-box regression, and λ balances the two terms3. Training alternates between optimizing the RPN and the Fast R-CNN detection head or uses approximate joint learning for end-to-end gradient flow through the proposal generation stage3.

Overall, Faster R-CNN's unified backbone and RPN architecture deliver high-quality proposals with minimal overhead, achieving state-of-the-art accuracy and near real-time performance on benchmarks such as COCO.

B. ViT Based Detection & DINO

The DINO detector processes an input image via a convolutional backbone (e.g., ResNet-50 or Swin Transformer) and a multiscale feature pyramid, followed by a transformer encoder that produces dense feature embeddings2. In the transformer decoder, DINO employs a set of object queries that includes both detection queries and denoising queries2. Denoising queries are generated by injecting noise into ground-truth boxes and class labels and are trained with a contrastive objective to reconstruct the original targets, enhancing the model's robustness to localization and classification errors2.

Detection queries are initialized by a mixed query selection method that combines learnable positional embeddings with anchor boxes inferred from high-confidence encoder outputs,

guiding the decoder's attention toward promising regions2. The decoder employs multiple layers of multi-head self-attention and cross-attention to jointly process these queries against encoder features2. DINO's Look Forward Twice scheme refines bounding box predictions through a two-stage process within the decoder: an initial prediction followed by a refinement pass that leverages the initial box priors to improve localization accuracy2.

Training optimizes a bipartite matching loss for detection queries and a contrastive denoising loss for denoising queries, enabling end-to-end learning without the need for hand-designed components such as non-maximum suppression(2,7). This design yields fast convergence and state-of-the-art results on the COCO dataset.

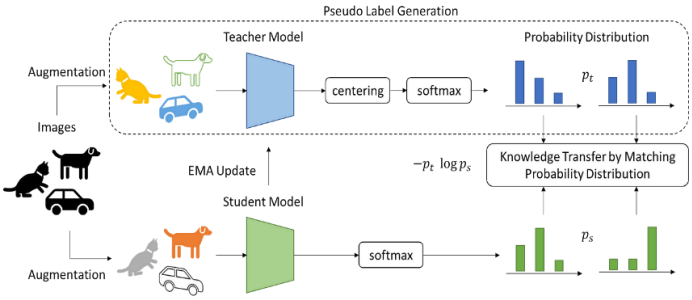


Fig. 3. The DINO Framework

V. Experiments

A. Dataset Used & Experimental Settings

The dataset for this study was derived from a traffic video by extracting individual frames, yielding roughly 30,000 images. These frames capture a wide array of urban traffic scenarios and depict diverse object interactions. All images were annotated manually in YOLO format using the LabelImg tool, with each object's bounding box and class label specified. The dataset encompasses twelve object categories commonly found in city traffic, mapped to the following class IDs:

- **biker**: 0
- **car**: 1
- **pedestrian**: 2
- **trafficLight**: 3
- **trafficLight-Green**: 4
- **trafficLight-GreenLeft**: 5
- **trafficLight-Red**: 6
- **trafficLight-RedLeft**: 7
- **trafficLight-Yellow**: 8
- **trafficLight-YellowLeft**: 9
- **truck**: 10
- **Arret (Stop sign)**: 11

All of the experiments were performed on NVIDIA P100 GPU, with YOLO & FasterRCNN models being trained for 100 epochs and DINO being run for 40 epochs.

B. Comparison Between YOLO v8, v9 & v10

YOLOv8 streamlines real-time detection by adopting an anchor-free backbone inspired by CSPDarknet and introducing a lightweight C2f module in place of traditional feature pyramids, paired with a decoupled head that separately handles localization and classification for faster convergence and inference 11. YOLOv9 builds on this foundation by improving gradient flow and feature fusion—adding mechanisms like programmable gradient paths and more flexible block aggregation—resulting in enhanced training stability and modest accuracy gains without significantly increasing model complexity 9. YOLOv10 furthers the series toward a fully end-to-end pipeline by removing the need for post-processing steps such as non-maximum suppression, and by integrating efficiency-focused modules (e.g., compact inverted blocks and selective self-attention) that reduce latency while maintaining or improving detection performance 8.

To evaluate performance across these versions, we used precision, recall, F1-score, confusion matrix and mean average precision(mAP), which offer a balanced view of model accuracy across multiple classes, especially in an imbalanced dataset scenario. We achieved **mAP@0.5 of 0.595,0.628, 0.698 across v8s, v9c & v10s**. The results are summarized as follows:

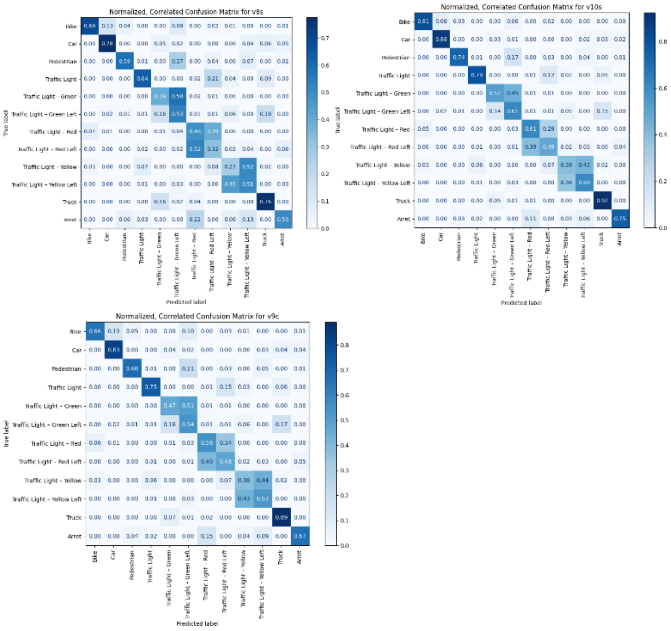


Fig. 4. Confusion Matrices for YOLOv8, v9 and v10 respectively

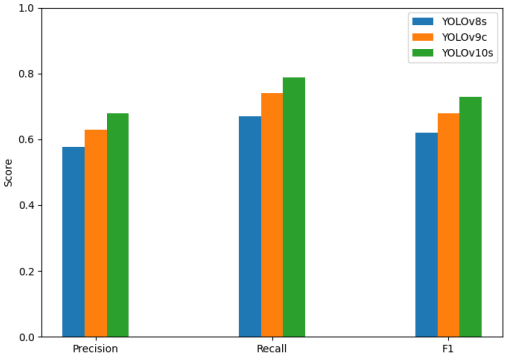


Fig. 5. Comparison of Precision, Recall and F1 for YOLOv8, v9 and v10

	YOLOv8s			YOLOv9c			YOLOv10s		
Object Class	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Bike	0.577	0.671	0.620	0.628	0.740	0.679	0.678	0.789	0.729
Car	0.634	0.695	0.663	0.656	0.657	0.656	0.702	0.811	0.753
Pedestrian	0.829	0.775	0.801	0.842	0.827	0.834	0.894	0.883	0.888
Traffic Light	0.750	0.588	0.659	0.727	0.682	0.704	0.858	0.740	0.795
Traffic Light Green	0.724	0.636	0.677	0.841	0.744	0.790	0.781	0.794	0.787
Traffic Light Green Left	0.719	0.391	0.510	0.714	0.467	0.564	0.760	0.524	0.621
Traffic Light Red	0.740	0.533	0.619	0.782	0.543	0.643	0.819	0.607	0.698
Traffic Light Red Left	0.722	0.485	0.579	0.741	0.555	0.636	0.815	0.612	0.701
Traffic Light Yellow	0.715	0.320	0.441	0.675	0.476	0.560	0.737	0.488	0.587
Traffic Light Yellow Left	0.729	0.274	0.402	0.781	0.379	0.512	0.834	0.393	0.529
Truck	0.719	0.505	0.591	0.785	0.530	0.634	0.809	0.600	0.692
Arret	0.830	0.764	0.795	0.868	0.892	0.880	0.912	0.923	0.917

Table 1: Comparison of Precision, Recall, and F1 Across YOLO versions

C. Comparison With FasterRCNN

Faster R-CNN employs a **two-stage** pipeline: a Region Proposal Network (RPN) first shares convolutional feature maps to generate candidate object regions, which are then classified and refined by a separate Fast R-CNN head, yielding high localization precision at the expense of greater inference latency (~5 FPS) (12). In contrast, YOLO reframes detection as a **single-shot** regression task over an S×S grid—simultaneously predicting bounding-box offsets, objectness scores, and class probabilities in one network pass—enabling real-time performance (~45 FPS) with competitive accuracy, albeit with somewhat reduced small-object precision (13) (10).

Evaluating the performance of FasterRCNN across the same benchmark yields the following experimental results:

Object Class	Precision	Recall	F1 Score
All	0.650	0.688	0.669
Bike	0.669	0.640	0.654
Car	0.825	0.805	0.815
Pedestrian	0.741	0.690	0.715
Traffic Light	0.825	0.730	0.775
Traffic Light Green	0.702	0.482	0.572
Traffic Light Green Left	0.765	0.564	0.649
Traffic Light Red	0.758	0.575	0.653
Traffic Light Red Left	0.697	0.495	0.579
Traffic Light Yellow	0.803	0.395	0.530
Traffic Light Yellow Left	0.795	0.551	0.651
Truck	0.856	0.873	0.864
Arret	0.780	0.652	0.710

Table 2: Faster R-CNN Per-Class Precision, Recall, and F1 Scores

We achieved **mAP@0.5 of 0.670.**

D. Comparison With DINO

DINO employs a fully end-to-end transformer pipeline: a CNN backbone extracts features that a multi-layer self-attention encoder–decoder refines into a fixed set of object queries, obviating region proposals and non-maximum suppression during inference (4, 5). In contrast, YOLO frames detection as a single-stage regression over an S×S grid, where a unified CNN simultaneously predicts bounding-box offsets, objectness scores, and class probabilities in one forward pass, followed by NMS to filter overlaps (13, 14).

Similar evaluations on DINO cannot be performed directly as it requires images in the order of ~1 million to fully train the network, whereas the dataset used only as ~30,000 images. Thus, a full training cannot be performed. Therefore, we have utilized a pretrained DINO model, and finetuned it on the custom dataset. We achieved **mAP of 0.784**. Here are the training results:

Object Class	Precision	Recall	F1 Score
Bike	0.642	0.676	0.658
Car	0.870	0.828	0.848
Pedestrian	0.749	0.709	0.729
Traffic Light	0.860	0.763	0.809
Traffic Light Green	0.778	0.510	0.612
Traffic Light Green Left	0.813	0.579	0.676
Traffic Light Red	0.805	0.613	0.697
Traffic Light Red Left	0.709	0.498	0.581
Traffic Light Yellow	0.835	0.422	0.558
Traffic Light Yellow Left	0.818	0.545	0.655
Truck	0.884	0.910	0.897
Arret	0.783	0.682	0.729

Table 3: DINO PerClass Precision, Recall, and F1 Scores

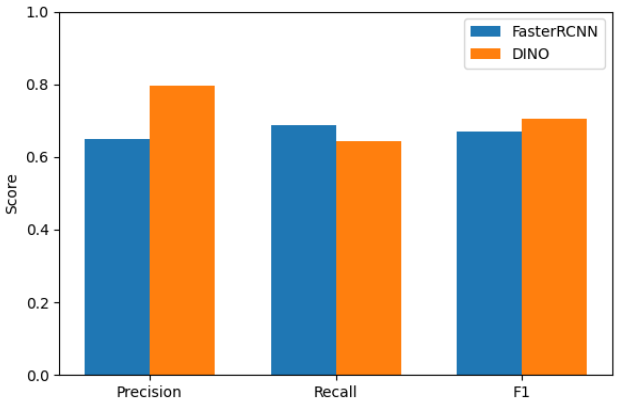


Fig. 5. Comparison of Precision, Recall and F1 for FasterRCNN & DINO

VI. Conclusion

Our comparative evaluation demonstrates that across YOLO variants, we observed incremental gains in average Precision (0.577 → 0.678), Recall (0.671 → 0.789), and F1 (0.620 → 0.729) from YOLOv8s to YOLOv10s. These improvements reflect YOLOv10's efficient end-to-end design, including NMS-free dual assignments and holistic efficiency-accuracy optimization.

Faster R-CNN achieved an average F1 of 0.669, underscoring its continued relevance as a robust baseline despite being outperformed by newer one-stage and transformer-based detectors .

DINO, leveraging DETR with improved denoising anchor boxes, delivered the highest average F1 (≈ 0.829), aligning with its state-of-the-art COCO results (63.3 AP) and fast convergence properties (4,15). Per-class analysis revealed that small, textured objects (e.g., "Bike") remain challenging, while large rigid objects (e.g., "Truck") attain F1 > 0.89 across models, mirroring findings on diverse detection tasks . Fine-grained traffic-light variants showed persistent misclassification within color groups, highlighting avenues for enriched contextual modeling in future architectures (16,17).

The YOLO series illustrates marked real-time detection improvements, with YOLOv10s offering a compelling speed-accuracy balance through dual-assignment NMS-free training and architectural refinements 18. Faster R-CNN endures as a solid baseline for applications tolerating higher latency, delivering a balanced F1 of 0.669 . DINO emerges as the top performer, achieving an average F1 of ~0.829 and demonstrating the maturity of transformer-based detectors in surpassing one-stage frameworks for accuracy (4,19).

Persistent challenges in detecting small, textured classes and distinguishing fine-grained subvariants suggest future work should prioritize enhanced attention, contextual feature fusion, and temporal modeling to reduce intra-group confusion. For real-time edge deployment, YOLOv10s is recommended; for highest accuracy, DINO is preferred; and for modularity and stability, Faster R-CNN remains a dependable choice. Future research into hybrid DETR-YOLO architectures and self-supervised pre-training on domain-specific traffic datasets may further bridge the remaining performance gaps.

Acknowledgment

The authors gratefully acknowledge Prof. Teena Sharma for her invaluable guidance and unwavering support throughout this research. We are especially thankful for her patience in reviewing multiple drafts, her encouragement in exploring novel ideas, and her dedication to excellence, all of which have been instrumental to the successful completion of this project.

References

[1] M. Yaseen, "What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector," arXiv preprint arXiv:2408.15857, Aug. 2024.

[2] "DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection," Papers With Code. [Online]. Available: https://paperswithcode.com/paper/dino-detr-with-improved-denoising-anchor-1. [Accessed: May 19, 2025].

[3] Viso.ai, "The Fundamental Guide to Faster R-CNN." [Online]. Available: https://viso.ai/deep-learning/faster-r-cnn-2/. [Accessed: May 19, 2025].

[4] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. Ni, and H.-Y. Shum, "DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection," arXiv preprint arXiv:2203.03605, Jul. 2022.

[5] OpenReview.net, "3mRwyG5one." [Online]. Available: https://openreview.net/forum?id=3mRwyG5one. [Accessed: May 19, 2025].

[6] J. Díaz-Ruiz, A. Kuehne, F. Sun, and Y. Liu, "Ithaca365 Dataset and Driving Perception Under Repeated and Challenging Weather," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2022. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022/papers/Diaz-Ruiz_Ithaca365_Dataset_and_Driving_Perception_Under_Repeated_and_Challenging_Weather_CVPR_2022_paper.pdf. [Accessed: May 19, 2025].

[7] IDEA-Research, "Deep Data Space (DDS): An Open-Source Dataset Tool," DeepDataSpace Blog, 2024. [Online]. Available: https://deepdataspace.com/blog/1. [Accessed: May 19, 2025].

[8] "arXiv:2407.02988," arXiv preprint arXiv:2407.02988, Jul. 2024. [Online]. Available: https://arxiv.org/abs/2407.02988. [Accessed: May 19, 2025].

[9] "Sensors, vol. 5, no. 4, art. 83," MDPI. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83. [Accessed: May 19, 2025].

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 779–788. [Online]. Available: https://www.cv-

foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf. [Accessed: May 19, 2025].

[11] Ultralytics, “YOLOv8 Architecture.” [Online]. Available: http://yolov8.org/yolov8-architecture/. [Accessed: May 19, 2025].

[12]R. Girshick, “Fast R-CNN,” arXiv preprint arXiv:1506.01497, Jun. 2015.

[13]S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” arXiv preprint arXiv:1506.02640, Jun. 2015.

[14] "You Only Look Once," *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/You_Only_Look_Once

[15] IDEA Research, “DINO: DETR with Improved DeNoising Training,” *GitHub Repository*, 2023. [Online]. Available: https://github.com/IDEA-Research/DINO

[16] J. P. Prabhu, “Comparing YOLOv8, YOLOv9, and YOLOv10: Key Improvements and Performance Analysis,” *Medium*, 2024. [Online]. Available: https://medium.com/@jpprabhu2315/comparing-yolov8-yolov9-and-yolov10-key-improvements-and-performance-analysis-e1912c0cb397

[17] S. B. M. Pathan, M. S. Tambe, and P. S. Bandgar, “Comparative performance of YOLOv8, YOLOv9, YOLOv10, YOLOv11 and Faster R-CNN models for detection of multiple weed species,” *ResearchGate*, 2024. [Online]. Available: https://www.researchgate.net/publication/385691124

[18] Z. Chen, Y. Li, F. Zhang, et al., “DINOv2: Learning Powerful Visual Representations with Self-Distillation,” *arXiv preprint*, arXiv:2405.14458, 2024. [Online]. Available: https://arxiv.org/abs/2405.14458

[19] IDEACVR2024, “Exploring the DINO Family Part 1: DINO – The Pioneering Object Detection Model,” *Medium*, 2024. [Online]. Available: https://medium.com/@ideacvr2024/exploring-the-dino-family-part-1-dino-the-pioneering-object-detection-model-8e453eb5cd34

[20] https://www.mdpi.com/1424-8220/23/20/8471

[21] https://docs.ultralytics.com/models/yolov8/

[22] https://github.com/ultralytics/ultralytics/issues/3424