Comparative Evaluation of Object Detection Models: YOLO, DETR, ViT-DETR, and OWL-ViT2

Mohammdkazem Rajabi
mohammadkazemr.rajabi@studenti.unipd.it
University of Padova

Abstract

Object detection remains a central challenge in computer vision, with growing interest in both high-performance closed-set models and flexible open-set systems. In this project, we conduct a comprehensive comparative evaluation of modern object detection architectures, focusing on their effectiveness across multiple configurations and generalization settings. Specifically, we investigate the performance of ***YOLOv8****[7],* ***DETR****[2], and* ***OWL-ViT2****[5] on the PASCAL VOC 2012 dataset[3].*

We fine-tune both convolutional and transformer-based models—including multiple YOLOv8 variants (e.g., nano, small, medium) and DETR models with different backbones such as ResNet and Vision Transformer—to assess how architectural choices and model scale impact detection accuracy, speed, and robustness. All models were trained using standardized data splits and augmentation pipelines to ensure fair comparisons.

In addition to closed-set evaluation, we explore open-vocabulary object detection using ***OWL-ViT2****, which leverages vision-language models to recognize previously unseen object categories without retraining. This enables a direct comparison between fine-tuned detectors and zero-shot models in terms of flexibility and real-world applicability.*

Each model was evaluated using standard metrics including mAP@0.5, precision, recall, and inference time, along with qualitative analysis under challenging detection scenarios.

Overall, this study offers actionable insights into the comparative strengths and limitations of CNN-based and transformer-based object detectors, guiding informed model selection for diverse real-world applications.

1. Introduction

1.1. Object Detection

Object detection is one of those core tasks in computer vision that's easy to describe—finding and labeling objects in images—but deceptively complex to do well. It's not just about recognizing what's in an image; it's about saying where each thing is, and doing it reliably, even when objects are small, overlapping, or half-hidden.

There are two main types of detectors people use today:

One-stage detectors, like YOLO or SSD, are built for speed. They predict object locations and classes in one go, making them ideal for real-time applications like self-driving cars or live video analysis.

In recent years, the game has shifted again with Transformer-based models. DETR, for example, uses the same architecture that revolutionized NLP and applies it to detection—treating object detection as a set prediction problem. It skips the hand-crafted parts like anchor boxes and post-processing, and just learns everything end-to-end. Vision Transformers (ViT) take this even further by treating image patches like tokens in a sentence and processing them with attention mechanisms, often resulting in strong performance, especially on complex or cluttered scenes.

2. Dataset

To train and evaluate our object detection we used the **PASCAL VOC 2012** dataset, downloaded in a curated version from Roboflow. This dataset is a standard benchmark in the object detection community and offers a diverse set of annotated real-world images suitable for training deep learning models.

2.1. Overview

The PASCAL VOC 2012 dataset consists of images annotated for object detection, where each object is described by its bounding box and class label. It includes **20 object categories**, such as *person*, *car*, *dog*, and *bicycle*, among others.

We used the following Roboflow-hosted version:

- **Dataset Source:** `https://universe.roboflow.com/jacob-solawetz/pascal-voc-2012/dataset/13`

- **Format:** Pascal VOC (XML annotations)
- **Total Images:** 17,000 (pre-split into train/val/test)
- **Number of Classes:** 20

2.2. Data Preparation

We structured the dataset using the typical VOC format, with images stored in `JPEGImages/`, annotations in `Annotations/`, and split definitions in `ImageSets/Main/`. Roboflow's export tools made it straightforward to convert the dataset into different formats depending on the model:

- **For YOLOv8:** The dataset was exported in YOLO format (TXT files with normalized coordinates). We used the Ultralytics YOLOv8 training interface, which automatically handled label parsing, image resizing, and augmentation.
- **For DETR:** We used the Pascal VOC (XML) format and created a custom PyTorch dataset class to parse XML annotations and convert them into the COCO-style format required by DETR. Images were resized dynamically while preserving aspect ratio, and data augmentations such as horizontal flipping and normalization were applied.

2.3. Class Mapping

We used a consistent mapping from class names to integer indices across both models. The background class is implicitly handled by the training framework and not included in the mapping:

2.4. Why PASCAL VOC 2012?

We selected this dataset for the following reasons:

- It is a well-known benchmark for object detection with extensive use in the literature.
- It provides clean annotations with multiple object categories per image.
- It is lightweight enough to allow fast experimentation while being diverse enough for evaluating generalization.
- Roboflow facilitates easy conversion into both YOLO and VOC/COCO-compatible formats, reducing the data preparation burden.

3. Model Architectures

This section provides a brief overview of the object detection architectures evaluated in this study: YOLOv8, DETR. These models represent three major paradigms in object detection: single-stage detectors, transformer-based methods, and two-stage region proposal frameworks.

3.1. YOLOv8

4. YOLO

Object detection is one of the core tasks in computer vision, involving the identification and localization of objects within an image. Traditional object detection methods such as R-CNN and Faster R-CNN [**?**]rely on multiple processing stages to identify and classify objects, making them computationally expensive.

YOLO (You Only Look Once) revolutionized object detection by treating it as a single regression problem. Unlike previous methods, YOLO applies a single neural network to the full image and directly predicts bounding boxes and class probabilities. This chapter provides a comprehensive explanation of YOLO's working mechanism, including its architecture, training methodology, loss function, advantages, limitations, and a comparison with other object detection models.

4.1. YOLO Architecture

Grid-Based Approach The input image is divided into an $S \times S$ grid. Each grid cell is responsible for detecting objects whose center falls within that cell. Each grid cell predicts:

- B bounding boxes, each defined by:

$$(x, y, w, h, c) \tag{1}$$

  where:

  - x, y - Center coordinates of the bounding box (relative to the grid cell).
  - w, h - Width and height of the bounding box (relative to the entire image).
  - c - Confidence score, defined as:

$$c = P(\text{Object}) \times IOU_{\text{pred, truth}} \tag{2}$$

- C class probabilities for each object category.

Thus, the final output tensor size is:

$$S \times S \times (B \times 5 + C) \tag{3}$$

4.1.1 Network Design

The YOLO network consists of two main parts:

- **Feature Extractor:** The first part is a deep convolutional neural network (CNN) that extracts features from the input image.
- **Detector:** The second part consists of fully connected layers that predict bounding boxes and class probabilities.

Convolutional Layers YOLO uses 24 convolutional layers inspired by the GoogLeNet architecture. Instead of inception modules , it uses 1×1 convolutions followed by 3×3 convolutions for dimensionality reduction. Fully Connected Layers After feature extraction, YOLO uses two fully connected layers to predict:

- Bounding box coordinates (x, y, w, h)
- Confidence scores (c)
- Class probabilities $(P(\text{Class}|\text{Object}))$

we can see the architecure

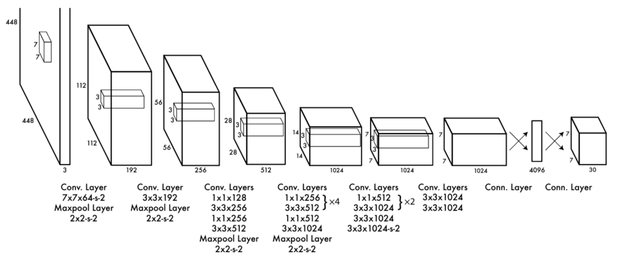
448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 1. Yolo architecture.

4.2. YOLO-NAS: Neural Architecture Search-Based Detection

YOLO-NAS (You Only Look Once with Neural Architecture Search) is a recent object detection model developed by Deci AI [1]. Unlike traditional YOLO variants, which are designed manually, YOLO-NAS leverages Neural Architecture Search (NAS) to automatically discover the most efficient architecture for object detection tasks.

This automated design process evaluates thousands of architectural configurations to find an optimal balance between accuracy, latency, and computational efficiency. As a result, YOLO-NAS achieves higher mAP and lower latency compared to earlier YOLO models like YOLOv5 or YOLOv8 [10].

In addition to architectural improvements, YOLO-NAS integrates post-training quantization and pruning techniques to optimize model size and inference speed, making it suitable for deployment on resource-constrained devices.

According to Roboflow's evaluation [9], YOLO-NAS models are available in various sizes (S, M, L), and surpass previous YOLO variants on benchmarks such as COCO and Pascal VOC, often without requiring Non-Maximum Suppression (NMS).

4.3. DETR

DETR[2] presents a paradigm shift by treating object detection as a direct set prediction problem. Instead of relying on heuristics such as anchor boxes, DETR employs a transformer-based architecture that predicts all objects in a single pass. This end-to-end approach eliminates the need for region proposals and post-processing techniques, making the model both conceptually simpler and more generalizable.

4.4. DETR Architecture

The DETR model is composed of three main components:

1. **CNN Backbone:** Extracts feature maps from the input image.
2. **Transformer Encoder-Decoder:** Processes and refines features using self-attention mechanisms.
3. **Prediction feed-forward networks (FFNs):** Produces the final object classes and bounding boxes.

4.4.1 CNN Backbone

The CNN backbone processes the input image $x_{\text{img}} \in \mathbb{R}^{3\times H_0 \times W_0}$, where H_0 and W_0 are the original height and width of the image. The backbone extracts feature representations and produces a lower-resolution feature map:

$$f \in \mathbb{R}^{C\times H\times W}, \tag{4}$$

where C is (typically 2048) and

$$H = \frac{H_0}{32}, \quad W = \frac{W_0}{32}. \tag{5}$$

This feature map is then passed into the transformer encoder.

4.5. Transformer Encoder

Before passing the feature map into the transformer encoder, a 1×1 convolution is applied to reduce the number of channels from C to a smaller dimension d. This results in a new feature map:

$$z_0 \in \mathbb{R}^{d\times H\times W}. \tag{6}$$

Since transformers operate on sequences rather than spatial feature maps, the spatial dimensions $H \times W$ are flattened into a single sequence of length HW, forming an input tensor of shape $d \times HW$.

4.6. Transformer Decoder

The transformer decoder in DETR follows the standard architecture of transformers and is responsible for converting the encoded image features into object predictions. Unlike conventional transformer decoders, which generate sequences in an autoregressive manner (one token at a time), DETR's decoder operates in parallel, predicting all objects simultaneously.

4.6.1 Parallel Decoding of Object Queries

The input to the decoder consists of:

- The feature representations output by the transformer encoder.
- A set of N learned embeddings, referred to as **object queries**.

Each object query is a fixed-length vector that represents a potential object in the image. These object queries are not associated with specific regions in the image beforehand; instead, the model learns to associate them with objects during training. Since the decoder operates on all N object queries simultaneously, it can leverage global relationships between all predicted objects.

4.6.2 Final Prediction

The output of the transformer decoder is a set of N learned embeddings, each of which corresponds to a potential object. These embeddings are then passed through a small prediction module consisting of:

- A feed-forward network (FFN) that predicts the class label using a softmax function.
- A separate linear layer that predicts bounding box coordinates in a normalized format.

Since DETR predicts a fixed number of objects, some object queries will not correspond to actual objects in the image. To handle this, an additional "no object" class is introduced. The model learns to assign unused queries to this category, allowing it to effectively filter out empty detections.

4.7. Comparison Between Facebook DETR and Roboflow RF-DETR

To understand the evolution of Transformer-based object detectors in practice, we compare the original DETR model from Facebook AI Research [2] with the recently released RF-DETR model by Roboflow [8]. While both models share the end-to-end detection formulation based on set prediction with Transformers, their design goals, usability, and performance characteristics differ substantially.

Architectural Core

Both models are inspired by the DETR paradigm: they frame object detection as a direct set prediction problem using a Transformer encoder-decoder and bipartite Hungarian matching loss. However, RF-DETR integrates refinements from later developments such as Deformable DETR [11] and Lightweight DETR (LW-DETR), and adopts a pre-trained DINOv2[6] ViT backbone. According to Roboflow's official release [8], "RF-DETR is the first real-time object detection model to surpass 60 mAP on COCO with no need for Non-Maximum Suppression (NMS)."

Training Pipeline and Usability

The original DETR codebase is designed for research and requires significant customization. It relies on COCO-style datasets, manual environment setup, and verbose data preprocessing. In contrast, RF-DETR wraps its training pipeline in a streamlined PyTorch Lightning implementation, supporting Roboflow JSON export format natively. This allows practitioners to "train a state-of-the-art object detector on custom datasets in minutes" [8].

Performance and Inference

While DETR is known for its elegant formulation and competitive performance, it suffers from slow convergence and large-scale training requirements. RF-DETR addresses this by adopting deformable attention modules and lightweight architectures. The RF-DETR Base model has 29M parameters, while the Large model contains 128M parameters. According to Roboflow, inference time on a T4 GPU is as low as 6ms per image, making it suitable for real-time deployment.

4.8. Architectural and Practical Differences Between DETR and RF-DETR

Although RF-DETR builds upon the original DETR framework [2], it introduces several targeted improvements that enhance efficiency, accuracy, and usability.

1. Deformable Attention Mechanism

RF-DETR replaces the global attention mechanism used in DETR with *deformable attention*, as proposed in Deformable DETR [11]. This approach allows each query to attend to a sparse set of key positions rather than the full feature map, leading to significantly faster convergence and improved performance on small and occluded objects.

2. Backbone Architecture

While the original DETR uses convolutional backbones such as ResNet-50[4] or ResNet-101, RF-DETR supports Vision Transformer (ViT) backbones, including options such as DINOv2 [6]. This enables RF-DETR to leverage powerful semantic representations learned from large-scale self-supervised pretraining, improving its ability to generalize across domains. According to Roboflow's official documentation [8], "RF-DETR is the first real-time object detection model to achieve over 60 mAP on COCO with no NMS, using a DINOv2 backbone and deformable attention."

3. Training and Usability

The original Facebook DETR implementation is designed for research and expects COCO-style datasets, requiring manual configuration and custom scripts. In contrast, RF-DETR simplifies training and deployment by using PyTorch Lightning and supporting Roboflow-formatted JSON datasets out of the box. This allows practitioners to train models on custom datasets in minutes with minimal setup, making it highly accessible for production use.

4. Real-Time Performance

RF-DETR is optimized for real-time inference. The base model contains only 29 million parameters and runs at approximately 6 milliseconds per image on an NVIDIA T4 GPU, while still achieving high detection accuracy. This makes RF-DETR suitable for deployment in latency-sensitive applications.
In summary, RF-DETR preserves the core idea of end-to-end set prediction from DETR but improves on it with deformable attention, ViT-based backbones, faster convergence, and ease of integration. These changes make RF-DETR a production-ready evolution of DETR that maintains accuracy while improving usability and efficiency.

5. Method

5.1. OWL-ViT2: Zero-Shot Object Detection

We implemented OWL-ViT2 using the HuggingFace Transformers library. Unlike traditional object detectors, OWL-ViT2 performs zero-shot detection, relying on natural language prompts instead of predefined class labels.

We used the `owlvit-base-patch32` model without any fine-tuning. A list of textual queries (e.g., "a car", "a person", "a building") was passed along with test images. The processor encoded both the image and text, and the model returned bounding boxes with confidence scores.

The following image shows an example detection result:

Although the confidence values were generally low, OWL-ViT2 correctly localized objects without any training on the dataset, validating its generalization capability in zero-shot mode.

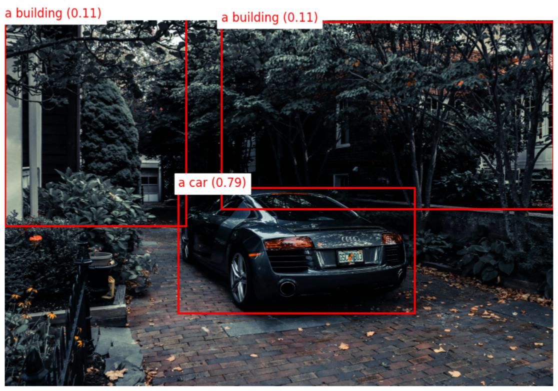

Figure 2. OWL-ViT2 zero-shot result showing detection of cars and buildings.

6. Fine-tuning

6.0.1 Fine-Tuning YOLO Using Ultralytics

We fine-tuned YOLOv8 using the Ultralytics framework, which supports training on custom datasets formatted in the YOLO annotation style. Our dataset, originally sourced in VOC format, was converted to YOLO format using Roboflow.

The fine-tuning process involved loading a pre-trained YOLOv8 model and replacing its classification head to match the number of classes in our dataset (20). This new head was randomly initialized, while the backbone and detection layers retained their pre-trained weights. The model was then trained on our dataset using the standard Ultralytics training pipeline.

This transfer learning approach allows the model to quickly adapt to new object categories while preserving the feature representations learned from large-scale datasets.

6.0.2 Fine-Tuning DETR on Pascal VOC

To adapt the DEtection TRansformer (DETR) model for our object detection task, we fine-tuned a pre-trained DETR model on the PASCAL VOC 2012 dataset. The original DETR model is trained on the COCO dataset with 91 classes, requiring modification of the classification head to match our dataset's 20 object categories plus one background class.

We replaced the original classification layer with a new randomly initialized head containing 21 output neurons. During training, this layer was learned from scratch, while the rest of the model retained its pre-trained weights. This allowed DETR to transfer its rich visual representations while adapting to the new domain.

A lower learning rate was used for the backbone and encoder-decoder layers, and a slightly higher rate for the new classification head, ensuring stable convergence. The model was trained using a combination of classification loss, bounding box regression loss, and generalized IoU loss. Standard data augmentations such as resizing, flipping, and normalization were applied to enhance generalization.

This fine-tuning strategy enabled DETR to effectively learn the object categories of the VOC dataset while maintaining its transformer-based detection capabilities.

7. Experiments

Discuss the experiments that you performed. The exact experiments will vary depending on the project, but you might compare with prior work, perform an ablation study to determine the impact of various components of your system, experiment with different hyperparameters or architectural choices. You should include graphs, tables, or other figures to illustrate your experimental results.

Table 1. YOLOv8 Fine-Tuning Results on VOC Dataset

Model	**mAP@50**	**Precision**	**Recall**
YOLOv8-N	0.70	71	60
YOLOv8-M	71	0.73	0.61
YOLOv8-L	0.74	0.77	0.64
YOLOv8-X	**0.76**	**0.79**	**0.67**

Table 2. Evaluation Metrics for DETR on Pascal

Model	**mAP@50**	**Average Recall**
DETR-ResNet50	0.720	0.710
DETR-ResNet101	**0.721**	**0.730**

Table 3. Fine-Tuned **RF-DETR** Results on VOC Dataset

Model	**mAP@50**	**Precision**	**Recall**
RF-DETR	0.74	0.75	0.72

Table 4. Fine-Tuned **yolo-nas** Results on VOC Dataset

Model	**mAP@50**	**Precision**	**Recall**
yolo-nas	0.68	0.69	0.62

8. Conclusion

In this report, we conducted a comprehensive comparative evaluation of several prominent object detection models, including YOLOv8, DETR, RF-DETR, and OWL-ViT2, using the PASCAL VOC 2012 dataset. Our experiments highlighted the strengths and trade-offs of each architecture in terms of accuracy, inference speed, model complexity, and generalization.

YOLOv8 models demonstrated excellent real-time performance with competitive mAP scores, particularly the larger variants (YOLOv8-L and YOLOv8-X), making them well-suited for speed-sensitive applications. **DETR**, while slower to converge, offered a robust end-to-end detection pipeline with strong performance on structured data. **RF-DETR** improved upon DETR by integrating deformable attention and Vision Transformer backbones, achieving better accuracy and inference time with easier usability.

We also explored **YOLO-NAS**, which applies neural architecture search to automatically find optimal detection architectures, outperforming traditional YOLO variants in both speed and accuracy. Finally, **OWL-ViT2** enabled zero-shot detection through vision-language alignment, demonstrating its ability to detect novel object categories without retraining, albeit with lower confidence scores.

Overall, our findings show that no single model universally outperforms others in all aspects. Transformer-based models like RF-DETR offer strong accuracy and flexibility, while models like YOLOv8 and YOLO-NAS provide fast, deployable solutions with minimal tuning. Zero-shot detection models like OWL-ViT2 open the door for more generalizable and language-driven object understanding, albeit with current limitations in precision.

References

[1] Deci AI. Yolo-nas: A new state-of-the-art for object detection, 2023.

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ECCV*, 2020.

[3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision (IJCV)*, 88(2):303–338, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[5] Matthias Minderer, Krishan Patil, Cordelia Schmid, Xiaohua Zhai, Konstantinos Drossos, Alexander Kolesnikov, Alexey Dosovitskiy, and Neil Houlsby. Simple open-vocabulary object detection with vision transformers. In *European Conference on Computer Vision (ECCV)*, 2022.

[6] Maxime Oquab, Théo Darcet, Theo Moutakanni, Alexandre Ramé, Ishan Misra, Julien Mairal, Piotr Bojanowski, Ivan Laptev, Cordelia Schmid, and Mathilde Caron. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 779–788, 2016.

[8] Peter Robicheaux, James Gallagher, Joseph Nelson, and Isaac Robinson. RF-DETR: A sota real-time object detection model. `https://blog.roboflow.com/rf-detr`, 2025.

[9] Roboflow. Yolo-nas: Next-generation object detection from deci, 2023.

[10] Ultralytics. Yolo-nas by deci – ultralytics docs. `https://docs.ultralytics.com/models/yolo-nas/`, 2023.

[11] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *ICLR*, 2021.