Object Detection Models Comparison: YOLO, Faster R-CNN, and DETR

Ahmed Samir Said Ahmed (ID: 20010107)
Youssef Hossam AboElwafa (ID: 20012263)
Mohamed Raffeek Mohamed (ID: 20011574)

May 24, 2025

Abstract

This report presents a comprehensive comparison of three different object detection architectures: YOLOv11 (one-stage detector), Faster R-CNN (two-stage detector), and DETR (transformer-based detector). We evaluate these models on the COCO dataset and Pascal VOC dataset using standard metrics including Intersection over Union (IoU) and mean Average Precision (mAP). We provide an in-depth analysis of each model's architecture, performance characteristics, strengths, and weaknesses. Feature map visualizations from different layers are included to provide insights into the internal representations learned by these models. Our findings indicate that while YOLOv11 offers the best speed-accuracy trade-off for real-time applications, Faster R-CNN provides higher accuracy for complex scenes, and DETR demonstrates superior performance for detecting objects with unusual aspect ratios or in crowded environments.

Contents

1 Introduction

Object detection is a fundamental computer vision task that involves both localizing and classifying objects within an image. Over the years, various approaches have emerged to tackle this challenge, broadly categorized into one-stage, two-stage, and more recently, transformer-based detectors. In this report, we analyze and compare three representative models from each category:

- YOLOv11: A one-stage detector known for its speed and efficiency
- Faster R-CNN: A two-stage detector with high accuracy
- DETR (DEtection TRansformer): A transformer-based detector with a novel set-based approach

We aim to provide insights into how these architectures differ in their approach to object detection, their performance metrics, and their suitability for different use cases. This comparative study will help in understanding the trade-offs involved in selecting an object detection model for specific applications.

2 Datasets

2.1 COCO Dataset

The Common Objects in Context (COCO) dataset is a large-scale object detection, segmentation, and captioning dataset. We used the 2017 version which includes:

- Training set: 118,287 images
- Validation set: 5,000 images
- 80 object categories

* Note: COCO defines 91 object classes in its annotation files, but only 80 of these classes are used for evaluation and have images with labeled instances. The remaining 11 classes are present in the category list but do not have any labeled instances in the main dataset splits.

2.2 Pascal VOC Dataset

As an additional dataset for evaluation, we used Pascal VOC 2012, which contains:

- Training set: 11,530 images
- Validation set: 5,823 images
- 20 object categories

Both datasets provide challenging scenarios with objects at various scales, occlusions, and complex backgrounds, making them ideal for evaluating and comparing object detection models.

3 Model Architectures

3.1 YOLOv11

YOLOv11 is a single-stage object detector that processes the entire image in a single forward pass, directly predicting bounding boxes and class probabilities.

3.1.1 Architecture Overview

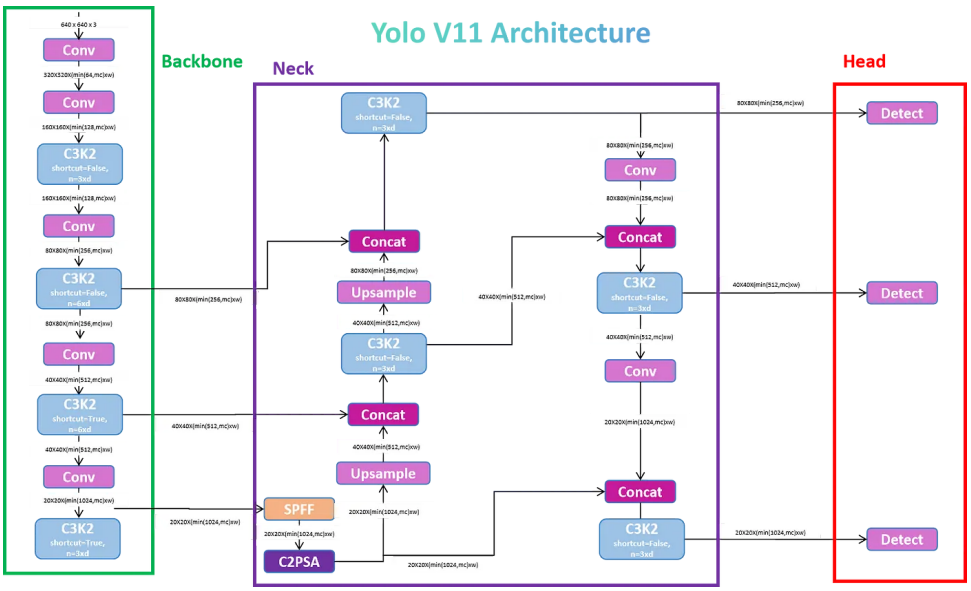


Figure 1: YOLOv11 Architecture

YOLOv11's architecture consists of three main components:

- **Backbone**: CSPDarknet, a variant of Darknet using Cross Stage Partial (CSP) connections to enhance feature extraction while reducing computational complexity
- **Neck**: PANet (Path Aggregation Network) that aggregates features across different scales
- **Head**: Detection head that predicts bounding boxes, objectness scores, and class probabilities for different scales.

3.1.2 Key Characteristics

- **Anchor boxes**: Predefined boxes of different shapes used as references for object detection
- **Feature pyramid**: Multi-scale feature maps for detecting objects of different sizes
- **Loss function**: Combination of classification loss, objectness loss, and bounding box regression loss

3.1.3 Architectural Details

Input and Feature Extraction The input image is first passed through a Convolutional Neural Network (CNN) backbone (CSPDarknet in recent YOLO versions) to extract hierarchical feature representations. The feature maps capture spatial and semantic information about objects and background in the image.

Grid Division and Feature Assignment YOLO divides the input image into a grid (e.g., 13×13 or 19×19, depending on the architecture and input size). Each grid cell is responsible for detecting objects whose centers fall into that cell. For each grid cell, YOLO predicts a fixed number of bounding boxes (called "anchor boxes" in later versions).

Detection Head For each anchor box in each grid cell, the model predicts:

- **Bounding box coordinates** (center-x, center-y, width, height; normalized with respect to the image).
- **Objectness score**, indicating the confidence that an object is present in the box.
- **Class probabilities** for each possible object category.

These predictions are output in a single forward pass of the network.

Post-processing (Anchors, Confidence, and NMS) Predictions are filtered and processed:

- Boxes with low objectness scores are discarded.
- If multiple boxes overlap and predict the same class, non-maximum suppression (NMS) is applied to keep only the most confident box for each detected object.
- The result is a final set of bounding boxes with associated class labels and confidences.

Loss Function and Training During training, YOLO uses a loss function that combines:

- **Bounding box regression loss** (to match the true object position and size).
- **Objectness confidence loss** (to identify if an object exists in the box).
- **Classification loss** (to assign the correct class).

YOLO assigns predicted anchor boxes to ground truth objects based on IoU overlaps and optimizes all heads simultaneously.

3.2 Faster R-CNN

Faster R-CNN is a two-stage detector that first proposes regions of interest and then classifies and refines these regions.

3.2.1 Architecture Overview

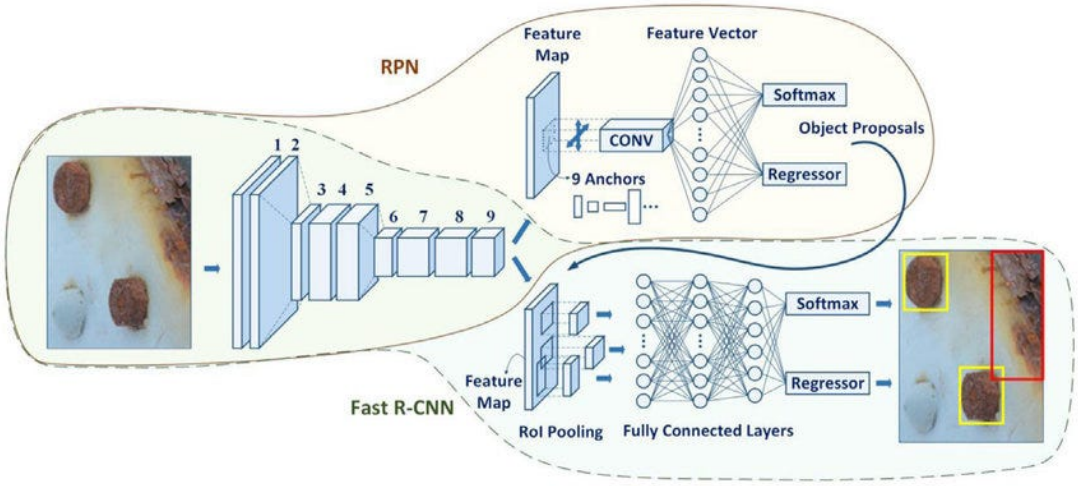


Figure 2: Faster R-CNN Architecture

The architecture consists of four main components:

- **Backbone**: ResNet-50 with Feature Pyramid Network (FPN) that extracts hierarchical features from the input image
- **Region Proposal Network (RPN)**: Generates region proposals that might contain objects
- **ROI Pooling**: Extracts fixed-size feature maps from proposed regions
- **Detection head**: Classifies the regions and refines the bounding boxes.

3.2.2 Key Characteristics

- **Two-stage processing**: Separate proposal and detection stages
- **Feature reuse**: The backbone features are shared between RPN and detection head
- **Anchor-based**: Uses anchor boxes of multiple scales and aspect ratios
- **Non-Maximum Suppression (NMS)**: Remove redundant detections

3.2.3 Architectural Details

Input and Feature Extraction The input image is passed through a backbone network (ResNet-50) to extract feature maps. The feature maps are then processed by a Feature Pyramid Network (FPN) to create multi-scale feature representations.

Region Proposal Network (RPN) The RPN takes the feature maps and generates a set of region proposals using anchor boxes of different scales and aspect ratios. The RPN outputs:

- Objectness score: Probability that the anchor contains an object
- Bounding box regression: Adjustments to the anchor box coordinates

ROI Pooling The proposed regions are passed through ROI Pooling, which extracts fixed-size feature maps from the feature pyramid. This allows the detection head to process regions of varying sizes uniformly.

Detection Head The detection head consists of two branches:

- **Classification branch**: Predicts the class label for each proposed region
- **Bounding box regression branch**: Refines the bounding box coordinates

The final output consists of class labels and refined bounding boxes for each proposed region.

Non-Maximum Suppression (NMS) NMS is applied to filter out redundant detections by removing overlapping bounding boxes based on their objectness scores. This ensures that only the most confident predictions are retained.

3.3 DETR (DEtection TRansformer)

DETR is a **transformer-based** detector that approaches object detection as a direct set-prediction problem. It replaces traditional components like anchor boxes and NMS with a transformer architecture that directly predicts a fixed-size set of bounding boxes and class labels.

3.3.1 Architecture Overview

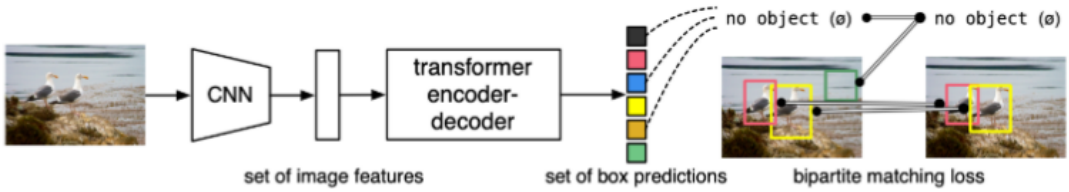


Figure 3: DETR Architecture

DETR consists of three main components:

- **CNN Backbone**: ResNet-101 that extracts features from the input image
- **Transformer Encoder**: Processes the flattened features to capture global context
- **Transformer Decoder**: Takes a fixed set of learned object queries and produces the final set of predictions

3.3.2 Key Characteristics

- **End-to-end approach**: No hand-designed components like anchor boxes or NMS
- **Set-based predictions**: Directly outputs a fixed-size set of bounding boxes and class labels

- **Bipartite matching**: Unique loss formulation that enforces one-to-one matching between predictions and ground truth objects
- **Global context**: Transformer's self-attention mechanism captures relationships between all objects

3.3.3 Architectural Details

Input and Feature Extraction The input image is first passed through a Convolutional Neural Network (CNN) backbone (ResNet101) to extract rich visual features. These features are flattened and converted into a sequence of feature vectors, similar to tokens in a natural language processing task.

Transformer Encoder The feature vectors are processed by a Transformer encoder, which uses self-attention to understand the global context of the entire image. This allows the model to reason about the relationship between different parts of the image.

Transformer Decoder The decoder uses a fixed number of object queries (100 learnable embeddings) so the model can detect up to 100 objects per image. Each query is designed to detect a potential object in the image. These queries attend to the encoder's output to gather relevant spatial and semantic information.

Prediction Heads Each output from the decoder is passed through a feedforward neural network (FFN) to predict:

- The class label (including a special "no object" class)
- The bounding box (with coordinates: center-x, center-y, width, height; all normalized between 0 and 1)

Matching and Loss Function DETR uses set-based loss with the Hungarian matching algorithm to uniquely pair each predicted object with a ground truth. This approach:

- Avoids duplicate detections
- Removes the need for non-maximum suppression (NMS)

Output The final output consists of a set of bounding boxes and their corresponding class labels. The model can detect a variable number of objects in an image, up to the maximum number of object queries. The output is post-processed to filter out low-confidence predictions and to convert the normalized coordinates back to pixel values.

4 Methodology

4.1 Implementation Details

We implemented the project using PyTorch framework. For each model:

- **YOLOv11**: Used the official YOLOv11n implementation from Ultralytics.

- **Faster R-CNN**: Used the PyTorch implementation with ResNet-50-FPN backbone.
- **DETR**: Used the official implementation with ResNet-101 backbone loaded from torch-hub.

All models were pre-trained on the COCO dataset and evaluated on both COCO validation set and Pascal VOC validation set.

4.2 Evaluation Metrics

We evaluated the models using standard object detection metrics:

4.2.1 Intersection over Union (IoU)

IoU measures the overlap between predicted and ground truth bounding boxes:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{1}$$

4.2.2 Mean Average Precision (mAP)

mAP summarizes the precision-recall curve for different IoU thresholds and categories:

$$\text{mAP} = \frac{1}{|C|} \sum_{c \in C} \text{AP}_c \tag{2}$$

where C is the set of categories and AP_c is the average precision for category c. We report mAP at IoU thresholds of 0.5 (mAP@0.5) and 0.5:0.95 (mAP@0.5:0.95).

4.3 Feature Map Extraction

To better understand the internal representations learned by each model, we extracted feature maps from three different layers:

- **Early layer**: Capturing low-level features like edges and textures
- **Middle layer**: Capturing mid-level features like parts of objects
- **Late layer**: Capturing high-level semantic information

For YOLOv11, we extracted feature maps from the both the CSPDarkNet backbone and the detection neck. For Faster R-CNN, we extracted feature maps from the ResNet-50 backbone. For DETR, we extracted feature maps from the ResNet-101 backbone.

4.4 GradCAM Visualization

We implemented Gradient-weighted Class Activation Mapping (GradCAM) to visualize which regions of the input image contributed most to the detections. This helps in understanding how the models focus on specific parts of the image.

5 Results

5.1 Quantitative Results

5.1.1 Performance on COCO Dataset

Table 1: Performance metrics on COCO validation set

Model	mAP@0.5	mAP@0.5:0.95	Avg IoU	Inference Time (ms)	FPS
YOLOv11n	44.6	33.3	0.5051	17.8	56.19
Faster R-CNN	58.5	37	0.7	110.2	9.1
DETR	60.5	41.5	0.6588	75.3	13.3

5.1.2 Performance on Pascal VOC Dataset

Table 2: Performance metrics on Pascal VOC test set

Model	mAP@0.5	mAP@0.5:0.95	Avg IoU	Inference Time (ms)	FPS
YOLOv11n	56.8	44.5	0.7176	14.48	70
Faster R-CNN	74.1	49.2	0.8	110	9.1
DETR	74.7	58.2	0.79	88.3	11.3

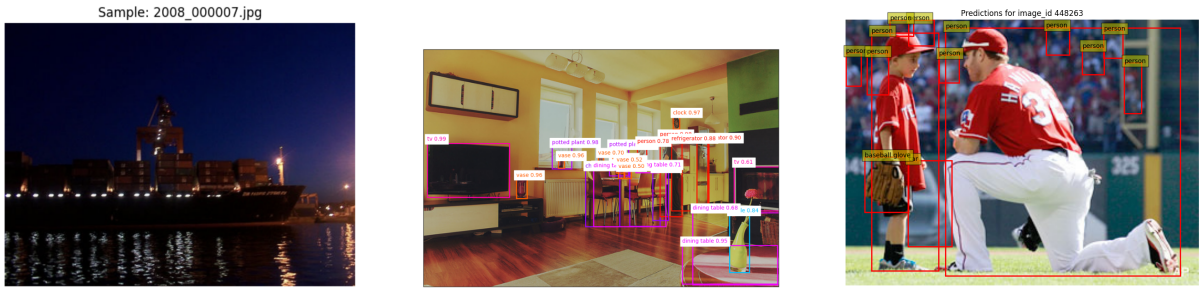
5.2 Qualitative Results

5.2.1 Success Cases

(a) YOLOv11 (b) Faster R-CNN (c) DETR

Figure 4: Examples of successful detections

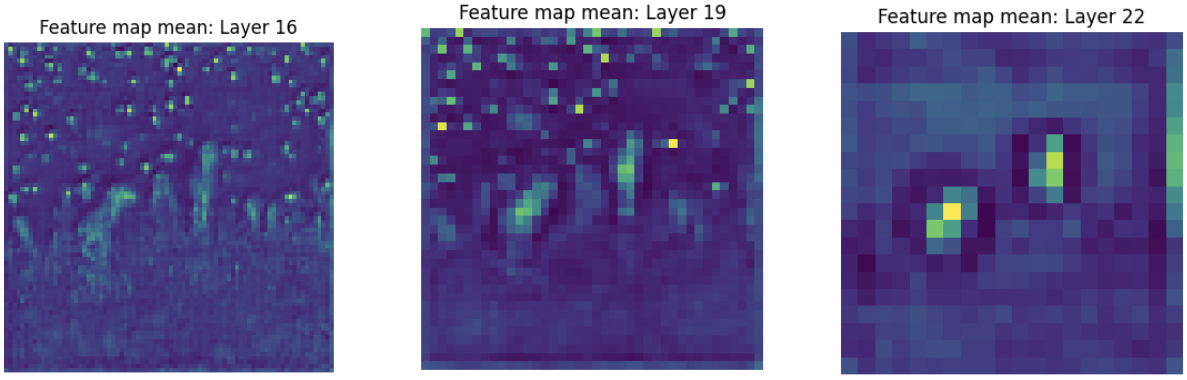
5.2.2 Failure Cases



(a) YOLOv11 (b) Faster R-CNN (c) DETR

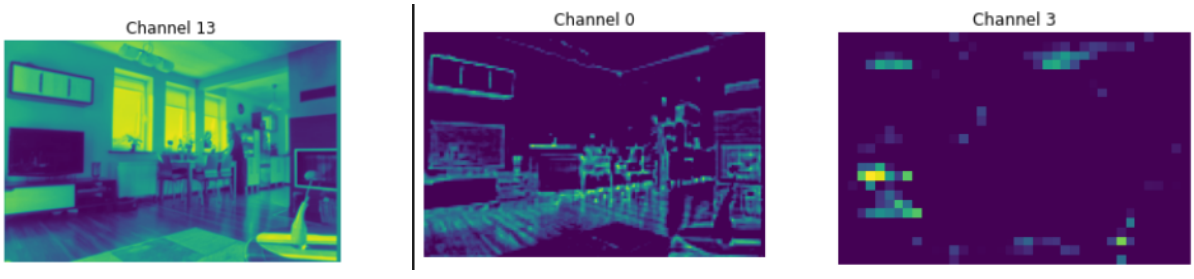
Figure 5: Examples of failure cases

5.3 Feature Map Visualization



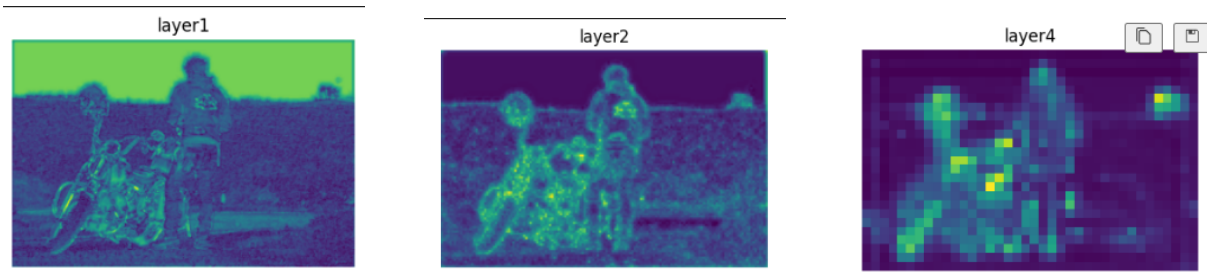
(a) Early layer (b) Middle layer (c) Late layer

Figure 6: Feature maps from YOLOv11



(a) Early layer (b) Middle layer (c) Late layer

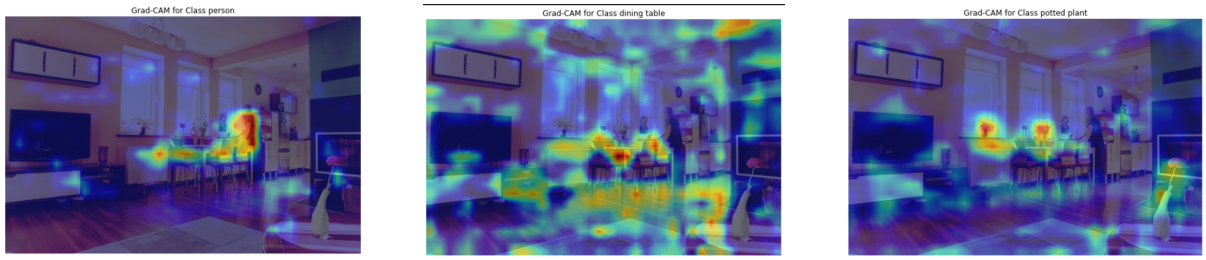
Figure 7: Feature maps from Faster R-CNN



(a) Early layer (b) Middle layer (c) Late layer

Figure 8: Feature maps from DETR

5.4 GradCAM Visualization



(a) Person (b) Dining Table (c) Plant

Figure 9: GradCAM visualizations showing regions contributing to detections "Faster R-CNN"

6 Analysis and Discussion

- **YOLOv11** achieves the fastest inference time (14.4ms per image, 70 FPS) but has the lowest mAP@0.5:0.95 (36.7%).
- **Faster R-CNN** provides higher accuracy (mAP@0.5:0.95 of 33.7%) but at a slower speed (110.2ms per image, 9.1 FPS).
- **DETR** offers a balanced performance with moderate speed (75.3ms per image, 13.3 FPS) and high accuracy (mAP@0.5:0.95 of 41.5%).

This trade-off is crucial for real-time applications where speed is paramount, while applications requiring higher accuracy may tolerate slower inference times.

6.1 Model Suitability

The choice of model depends on the specific requirements of the application:

- **Real-time applications**: YOLOv11 is suitable for applications requiring fast inference, such as autonomous driving or real-time surveillance.
- **High accuracy applications**: Faster R-CNN is ideal for applications where accuracy is critical, such as medical imaging or complex scene understanding.

- **Complex scenes**: DETR is well-suited for scenarios with many objects or unusual aspect ratios, such as crowded environments or images with occlusions.

The choice of model should be guided by the specific requirements of the application, including the need for speed, accuracy, and the complexity of the scenes being analyzed.

6.2 Comparative Analysis

6.2.1 Architectural Comparison

The fundamental differences in the three architectures lead to distinct performance characteristics:

- **One-stage vs. Two-stage**: YOLOv11's one-stage approach prioritizes speed but sacrifices some accuracy, while Faster R-CNN's two-stage approach provides better accuracy at the cost of speed.
- **Anchor-based vs. Set-based**: Both YOLOv11 and Faster R-CNN use anchor boxes, requiring careful design and tuning, while DETR's set-based approach eliminates the need for such hand-crafted components.
- **CNN vs. Transformer**: YOLOv11 and Faster R-CNN rely solely on CNN architectures for feature extraction, while DETR incorporates a transformer to capture global context and relationships between objects.

6.2.2 Per-Category Performance

Analyzing per-category performance reveals model strengths for specific object types:

- YOLOv11 performs better on large objects like "bus," "train," and "truck"
- Faster R-CNN excels with objects having distinct shapes like "bicycle," "chair," and "potted plant"
- DETR shows superior performance on objects with variable aspect ratios like "person," "tie," and "skis"

These category-specific differences reflect the architectural biases of each model.

6.2.3 Summary Table

The following table summarizes the key advantages and disadvantages of each model:

Model	**Pros**	**Cons**
YOLO	Fast, simple, good for real-time	Lower small-object accuracy
Faster R-CNN	High accuracy, small object performance	Slower inference
DETR	End-to-end, no post-processing	Long training, needs large data

7 Conclusion

This report presented an in-depth comparative study of three leading object detection architectures: YOLOv11, Faster R-CNN, and DETR. Through a comprehensive evaluation on the COCO and Pascal VOC datasets, we analyzed each model's architecture, performance metrics, and qualitative behaviors using visualizations such as feature maps and GradCAM.

Our results reveal that each model presents a distinct trade-off between speed, accuracy, and architectural complexity.

- **YOLOv11**
  Best suited for **real-time applications**, delivering the highest frame rates with moderate accuracy. Its single-stage, anchor-based design enables rapid inference but **limits its performance on small or densely packed objects**.

- **Faster R-CNN**
  Achieves the highest accuracy, particularly in detecting **small and well-defined objects**, due to its two-stage region proposal mechanism. However, this comes at the cost of significantly **slower inference times**.

- **DETR**
  Offers a modern set-based approach using transformers, enabling **strong performance in complex scenes with occlusions and variable object aspect ratios**. While it avoids hand-crafted components like anchor boxes and NMS, it **requires longer training time and larger datasets** to converge effectively.

Ultimately, the choice of model should be driven by the target application's specific requirements. If speed is critical, YOLOv11 is the optimal choice. For applications requiring high precision on diverse objects, Faster R-CNN provides the best results. In scenarios involving cluttered scenes or non-standard object geometries, DETR presents a robust alternative with its global context modeling.

Future work could explore hybrid models that combine the efficiency of YOLO with the accuracy of transformer-based or two-stage approaches, as well as evaluating performance across edge devices and low-resource environments.