Fine-Tuning YOLO, DETR, and Faster R-CNN for Object Detection on PASCAL VOC 2012 Dataset

MohammadKazem Rajabi

1. Object Detection

1.1. Introduction

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.

Two-stage detectors, like Faster R-CNN, take a bit more time but usually get better accuracy. They first generate possible object regions, then refine them and classify what's inside. It's like looking twice to be sure.

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. Model Evaluation

Evaluating model performance in computer vision tasks requires specific metrics to quantify accuracy. The most commonly used evaluation metric for object detection is *mean average precision* (mAP). This metric is calculated for both bounding boxes (mAP_{box}) and segmentation masks (mAP_{mask}) in the case of instance segmentation models. Leading object detection frameworks such as YOLO and DETR rely on mAP to assess model accuracy and performance in research publications.

The effectiveness of mAP lies in its suitability for detecting multiple objects within an image, ensuring that all objects of interest are considered. Additionally, it incorporates both precision and recall, balancing false positives (FP) and false negatives (FN) to provide a more comprehensive measure of detection accuracy. This section provides an in-depth explanation of mAP computation, detailing its application in evaluating the model used in this research.

2.1. Mean Average Precision

The **mean average precision (mAP)** is computed as the mean of individual class-wise average precision (AP) values, which are derived from recall and precision scores. These values range between 0 and 1. The methodology for computing mAP is outlined below, focusing on object detection (bounding boxes). The same principles apply to segmentation tasks.

- **Intersection over Union (IoU)**: The first step in evaluating prediction accuracy is computing the IoU score, which measures the overlap between the predicted and actual object bounding boxes. A prediction is considered correct (true positive) if its IoU with the ground truth exceeds a predefined threshold (e.g., 50%, 75%, 90%). This threshold determines the strictness of the mAP calculation. Predicted bounding boxes are sorted in descending order based on confidence scores, and those below a certain threshold are discarded from evaluation. Figure 1 illustrates the IoU computation.

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

- **Precision and Recall**: These two key metrics are computed for each object class as follows:

$$\begin{cases} \text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \\ \text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \end{cases} \tag{2}$$

  where:

  - True Positives (TP) refer to correctly detected objects (IoU above the threshold).
  - False Positives (FP) represent incorrect detections that do not match any ground truth.

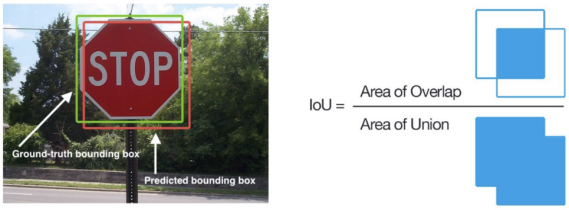

Figure 1. Illustration of the Intersection over Union (IoU) metric, which quantifies bounding box prediction accuracy.

  - False Negatives (FN) are ground truth objects that the model fails to detect.

- **Precision-Recall Curve**: Precision-recall curves are generated for different threshold values. The area under these curves corresponds to the average precision (AP) for each class. The calculation uses the trapezoidal rule:

$$\text{AP}^{IoU} = \int_0^1 p(r)dr \tag{3}$$

where $p(r)$ represents the interpolated precision at recall r.

- **Average Precision (AP)**: The mean of AP values is computed across all N classes to obtain a final score for each IoU threshold. The IoU thresholds range from 0.1 to 0.95, with increments of 0.05, resulting in 17 equally spaced intervals. The formula is given by:

$$\text{mAP}^{IoU} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i^{IoU}, \qquad \text{where} \quad IoU \in [0.1, 0.95] \tag{4}$$

- **Final mAP Computation**: The overall model performance is given by the mean of all mAP^{IoU} values.

To account for different levels of strictness in object detection, mAP is often computed at multiple IoU thresholds, leading to variations such as AP^{50} and AP^{75}. Specifically: - AP^{50}represents the AP value when the IoU threshold is at least 0.5. - AP^{75} is a stricter metric, requiring at least 75% overlap for a true positive detection.

Since evaluating mAP on large datasets at every step of training can be computationally expensive, it is typically calculated at specific checkpoints (e.g., every n training epochs) to track model progress.

The Intersection over Union (IoU) metric used for evaluating bounding box accuracy is illustrated in Figure 1

3. Dataset

3.1. Overview

For the experiments in this thesis, we used the **PASCAL VOC 2012** dataset hosted on Roboflow. This version contains a total of **17,112 images**, significantly more than the original PASCAL VOC 2012 dataset (which includes 11,530 train/val images). Each image is annotated using the standard VOC XML format and includes bounding boxes for one or more object instances, labeled with one of 20 predefined classes.

We selected this dataset for its balance between quality, availability, and compatibility with multiple object detection frameworks. It served as the benchmark for evaluating YOLOv5, DETR, and Faster R-CNN in our experiments.

3.2. Dataset Composition

- **Total Images:** 17,112
- **Annotation Format:** PASCAL VOC XML
- **Image Format:** JPEG
- **Number of Classes:** 20
- **Average Objects per Image:** Approximately 2.3

Each annotation file includes the following information:

- Image metadata: filename, width, height, depth
- Object-level tags:
  - `<name>`: class label (e.g., `person`)
  - `<bndbox>`: bounding box with `xmin`, `ymin`, `xmax`, `ymax`

3.3. Object Classes

The dataset includes the following 20 object categories:

```
aeroplane, bicycle, bird, boat,
bottle, bus, car, cat, chair,
cow, diningtable, dog, horse,
motorbike, person, pottedplant,
sheep, sofa, train, tvmonitor
```

These classes span multiple domains, including animals, vehicles, furniture, and people, offering a diverse challenge for detection models.

3.4. Preprocessing and Splits

The dataset was processed and exported using Roboflow's built-in tools. We applied the following steps to prepare the data for training:

- Resized all images to:
  - **640×640** for YOLOv5
  - **800×800** for DETR
- Converted annotations to the following formats:

  - YOLO format: normalized `class x_center y_center width height`
  - COCO-style JSON for DETR compatibility
  - VOC XML (original format) used with Faster R-CNN
- Performed an 80/20 split for training and test sets
- Reserved 10% of the training set for validation

3.5. Data Augmentation

To improve generalization and robustness, we applied the following augmentations during training:

- Random horizontal flipping
- Color jittering
- Random scaling and cropping
- Gaussian noise

These transformations were implemented using library-specific augmentations (e.g., Albumentations for YOLO, torchvision for Faster R-CNN, and custom augmentations for DETR).

3.6. Annotation Quality and Format Compatibility

The original PASCAL VOC annotations are known for their precision and consistency. The Roboflow-hosted version provided additional benefits:

- Cloud preprocessing and augmentation pipeline
- Export support for multiple formats (YOLO, COCO, VOC)
- Consistent resolution and class label formatting
- Dataset versioning for reproducibility

3.7. Limitations

Despite its popularity, the VOC dataset has some inherent limitations:

- No segmentation masks (only bounding boxes)
- Smaller image sizes compared to more modern datasets like COCO
- Uneven class distribution (e.g., "person" occurs more frequently than "sheep" or "train")

Nonetheless, due to its clean annotations, accessible formats, and compatibility with multiple model architectures, the PASCAL VOC 2012 dataset remains an excellent choice for controlled experimentation in object detection.

4. Model Architectures

In this section, we outline the object detection models used in our experiments. Each model represents a different architectural philosophy, allowing us to compare performance across paradigms.

- **YOLOv5**
- **DETR (DEtection TRansformer)**
- **Faster R-CNN**
- **CLIP**

4.1. Overview of CLIP's Architecture

4.1.1 Modifications from ConVIRT in CLIP

While CLIP builds upon the contrastive learning approach established in ConVIRT [1], several key modifications were introduced to improve training efficiency and simplify implementation:

- **Training from Scratch:** Unlike ConVIRT [1], which utilizes pre-trained weights for initialization, CLIP is trained entirely from scratch, avoiding dependencies on ImageNet or pre-trained text encoders.
- **Linear Projection Instead of Non-Linear:** CLIP simplifies the projection process by using a linear mapping from encoder representations to the shared embedding space, removing the non-linear projection introduced by Bachman et al. (2019) [2].
- **Simplified Text Preprocessing:** The text transformation function from ConVIRT, which sampled a single sentence from longer text, was removed in CLIP since most image-text pairs in its dataset consist of single-sentence descriptions.
- **Minimal Image Transformations:** CLIP reduces data augmentation complexity by applying only a random square crop from resized images, in contrast to ConVIRT's more extensive image transformations.

A new dataset comprising 400 million image–text pairs was constructed to train a simplified variant of ConVIRT from scratch. This approach, named CLIP (Contrastive Language-Image Pre-training), demonstrates the effectiveness of learning visual representations directly from natural language supervision.

CLIP consists of two main components:

1. An Image Encoder: Processes visual input and extracts high-dimensional representations.
2. A Text Encoder: Converts textual descriptions into a comparable embedding space.

Both encoders are trained jointly to align their representations in a shared multi-modal space, allowing the model to determine the similarity between an image and a textual description.

4.1.2 Image Encoder

CLIP's image encoder is based on either:

- A ResNet [3] architecture with modifications for enhanced feature extraction.
- A Vision Transformer (ViT) [**?**], which captures long-range dependencies within an image, improving generalization across diverse object categories.

Regardless of the backbone, the image encoder outputs a feature vector, mapping the visual input into a shared representation space.

4.1.3 Text Encoder

The text encoder in CLIP is a Transformer [4]-based model. It processes textual prompts (e.g., "a photo of a car") and generates semantic embeddings that align with corresponding visual representations.

The key advantage of this approach is that it does not require manually defined categories; instead, it learns relationships between text and images in a flexible, open-ended manner.

4.1.4 Image-Text Alignment and Embedding Space

CLIP operates by aligning images and text in a common feature space. During training, it learns to associate an image with its correct textual description while pushing apart mismatched pairs. The result is a multi-modal embedding space, where visually similar objects are closer to their corresponding textual descriptions.

How CLIP Matches Images to Text:

1. The image and text encoders produce feature vectors for both modalities.
2. These feature vectors are normalized and projected into the same dimensional space.
3. The model computes the cosine similarity between the image and multiple textual embeddings.
4. The textual description with the highest similarity score is considered the best match.

This process allows CLIP to perform classification tasks without explicitly training on specific object categories, making it highly effective for zero-shot learning.

An overview of the CLIP training and inference pipeline is shown in Figure 2.

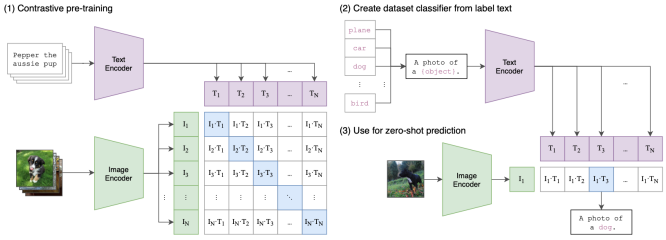

Figure 2. Unlike conventional image models that train an image encoder alongside a linear classifier to predict predefined labels, CLIP trains both an image encoder and a text encoder simultaneously to learn associations between images and their corresponding textual descriptions. During inference, the text encoder generates a zero-shot classifier by embedding the class names or descriptions from the target dataset.The figure is taken from [5]

4.1.5 Contrastive Learning for Visual Representation: The ConVIRT Framework

4.1.6 Introduction to ConVIRT

Contrastive Visual Representation Learning from Text (ConVIRT) [1] is a framework designed to enhance visual representation learning by leveraging the natural pairing of images and textual descriptions. It builds on the principle that text provides rich contextual information, which can serve as a powerful supervisory signal for training visual models. By maximizing the similarity between true image-text pairs and minimizing the similarity between incorrect pairs, ConVIRT enables more effective representation learning.

ConVIRT introduces a contrastive learning approach that allows models to align images with their corresponding text while ensuring that non-matching pairs remain distinct. Unlike traditional supervised learning that requires explicit labels, this framework takes advantage of existing textual descriptions, making it highly scalable. Since its introduction, ConVIRT has influenced several large-scale vision-language models, including CLIP, which utilizes a simplified version of its contrastive learning methodology.

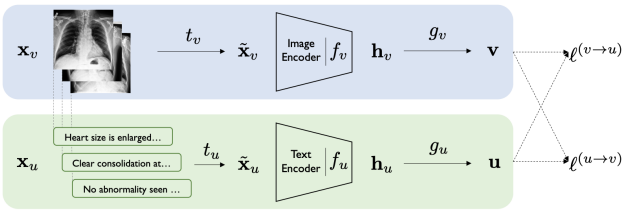

Figure 3. The blue and green shades represent the image and text encoding pipelines, respectively. Our method relies on maximizing the agreement between the true image-text representation pairs with bidirectional losses $\ell^{(v\to u)}$ and $\ell^{(u\to v)}$. The figure is taken from [1]

4.1.7 Task Definition

The core objective of ConVIRT is to learn visual representations by mapping images to a high-dimensional feature space where they maintain semantic relationships with textual descriptions. The learning process assumes paired input data, where each image is accompanied by a corresponding textual description. The goal is to develop an image encoder that transforms input images into fixed-dimensional vectors, capturing meaningful features that can be used for downstream applications such as classification and retrieval.

Each input sample consists of:

- x_v: An image or a set of images representing visual information.
- x_u: A text sequence describing the content of x_v.

The objective is to train an encoder function f_v, which maps an image x_v into a compact vector representation. This encoder is modeled as a convolutional neural network (CNN), allowing it to extract visual features efficiently. The learned image representations can then be transferred to downstream tasks, enabling flexible and efficient applications.

The use of paired image-text data is particularly advantageous, as such associations naturally occur in many real-world contexts. By learning from these image-text pairings, ConVIRT captures meaningful relationships between modalities, leading to improved generalization in vision-based tasks.

4.1.8 Contrastive Learning with Image-Text Pairs

The ConVIRT framework applies a contrastive learning paradigm, where images and their corresponding text descriptions are embedded into a shared vector space. The fundamental idea is to train the model such that representations of correct image-text pairs are close together, while representations of mismatched pairs are pushed apart. This learning strategy enables the model to develop a strong cross-modal understanding.

To achieve this, both images and text undergo a transformation process before being fed into their respective encoders:

- For images, a transformation function applies random augmentations such as cropping, flipping, and color jittering to introduce variability and prevent overfitting.
- For text, a span of words is randomly sampled from the full description to provide a more generalized representation.

The encoded representations are then projected into a common embedding space using learnable transformation functions. At this stage, the similarity between image and text embeddings is computed using cosine similarity, and a contrastive loss function is applied to optimize the model. The objective is to ensure that the model correctly associates images with their textual descriptions while discouraging false pairings.

4.1.9 Bidirectional Contrastive Learning

Unlike conventional contrastive learning methods that operate within a single modality, ConVIRT employs a bidirectional contrastive objective. This means the model is trained to:

- Align image representations with their corresponding text descriptions.
- Align text representations with their corresponding images.

This bidirectional learning strategy is crucial for improving the model's ability to retrieve relevant visual content given a text query, and vice versa. It also ensures that both modalities contribute equally to the representation space, leading to better generalization across different tasks.

The training process involves a contrastive loss function that enforces alignment between matched image-text pairs while penalizing incorrect pairings. This loss function is asymmetric for each modality, ensuring that the model learns meaningful associations across both image and text domains.

4.1.10 Realization of the ConVIRT Framework

ConVIRT is designed to be flexible in its implementation, allowing for various choices of encoders and transformation functions. The framework consists of the following key components:

- **Image Encoder:** A convolutional neural network (CNN), specifically a ResNet-50 [3]architecture, is used to encode images into feature representations.
- **Text Encoder:** A transformer-based model, is used to encode text into vector representations. The text encoder is initialized with pre-trained weights and fine-tuned for contrastive learning.
- **Projection Functions:** Separate neural networks with single-hidden-layer architectures are used to map image and text representations into a common embedding space.

To improve generalization, the text encoder is partially frozen during training, with only the last few transformer layers being updated. This strategy ensures that the model retains its pre-trained knowledge while adapting to the contrastive learning objective.

Additionally, data augmentation plays a crucial role in training. Image transformations include cropping, flipping, affine transformations, and color jittering, which introduce variability and improve robustness. For textual data, random sentence sampling is applied to enhance diversity while preserving semantic meaning.

By leveraging these techniques, ConVIRT creates a shared embedding space where image-text relationships are effectively learned and optimized for downstream tasks.

4.1.11 Optimization Strategy

The optimization process in ConVIRT is centered around a contrastive loss function that enhances alignment between image and text representations. The model learns to:

- Increase similarity scores between correctly matched image-text pairs.
- Decrease similarity scores between mismatched pairs.

To achieve this, a bidirectional loss function is employed, incorporating: where h_{v_i}, u_i represents the cosine similarity

$$h_v, u_i = \frac{v \cdot u}{\|v\|\|u\|}$$

- **Image-to-Text Contrastive Loss:** Ensures that an image representation aligns with its correct textual description.

$$\ell_i^{(u \to v)} = -\log \frac{\exp\left(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{u}_i, \mathbf{v}_k \rangle / \tau\right)} \tag{5}$$

- **Text-to-Image Contrastive Loss:** Ensures that a text representation aligns with its correct image.

$$\ell_i^{(v \to u)} = -\log \frac{\exp\left(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{v}_i, \mathbf{u}_k \rangle / \tau\right)} \tag{6}$$

The final optimization function is computed as a weighted combination of these loss functions, ensuring balanced learning across modalities. This approach results in a robust model that excels in multi-modal retrieval and zero-shot learning tasks.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\lambda \ell_i^{(v \to u)} + (1 - \lambda) \ell_i^{(u \to v)} \right) \tag{7}$$

4.2. Vision Transformer

The Vision Transformer (ViT) [6] is a relatively recent innovation in computer vision that presents a compelling alternative to traditional Convolutional Neural Networks (CNNs). Unlike CNNs, which rely on hierarchical feature extraction through convolutional operations, ViT is built upon the Transformer model—an architecture that has revolutionized natural language processing (NLP) and set new benchmarks in various domains.

One of the key advantages of ViT lies in its ability to process images efficiently through self-attention mechanisms rather than convolutional layers. This approach allows for a more global understanding of an image, capturing long-range dependencies between different regions. While ViT has demonstrated superior computational efficiency compared to CNNs, it comes with a significant drawback: it demands large-scale datasets for effective training and generalization. Unlike CNNs, which benefit from strong inductive biases that help them learn efficiently from smaller datasets, ViTs require extensive pretraining on massive datasets to achieve competitive performance.

4.2.1 Model Architecture

The **Vision Transformer (ViT)** follows the standard Transformer architecture as closely as possible while adapting it for **computer vision tasks**. The key difference from traditional convolutional neural networks (CNNs) is that ViT processes images **globally**, rather than through localized feature extraction. The architecture consists of four primary stages:

- **Converting 2D Images into 1D Patch Embeddings** Instead of feeding an entire image into the model as a whole, ViT **splits the image into smaller, fixed-size patches**. These patches are then **flattened into 1D vectors** before being passed to the Transformer. The size and number of patches depend on the image resolution and the chosen patch size, both of which act as hyperparameters. This conversion step allows the Transformer to process image data in a structured format, much like words in an NLP task.
- **Adding a Classification Token & Positional Embeddings**

Since Transformers **do not inherently understand spatial relationships**, additional information must be provided. To do this:

  - A **classification token** is prepended to the sequence of patches, acting as a global representation of the entire image.
  - **Positional embeddings** are incorporated to retain spatial information, ensuring that the model knows the relative positioning of each patch.

- **Processing Through the Transformer Encoder**
  The sequence of patch embeddings, along with the classification token, is passed through multiple layers of the **Transformer Encoder**. Each layer applies:

  - **Multi-Head Self-Attention (MSA)**, allowing patches to exchange information across the entire image.
  - **Feedforward Neural Networks (MLPs)** to refine features.
  - **Layer Normalization** and **Residual Connections**, which stabilize training and improve performance.

- **Final Prediction Layers**
  The **classification token** output from the last Transformer layer is used to generate predictions. For **classification tasks**, a **simple MLP head** is added on top. In contrast, for **object detection and segmentation**, more complex structures (e.g., detection heads like those in DETR) are integrated into the framework.

An overview of the Vision Transformer architecture is shown in Figure 4.

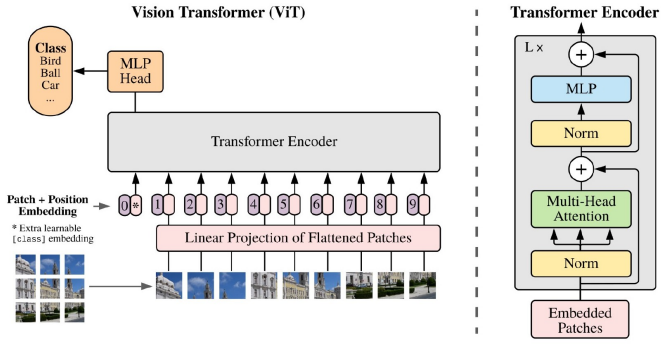

Figure 4. Vision Transformer model overview The illustrative picture is taken from [6]

4.2.2 Mathematical Representation of ViT Processing

To mathematically formalize the process, let's define an input image x with resolution $H \times W$ and C color channels (e.g., RGB images have $C = 3$):

- **Patch Extraction**
  The image is divided into N non-overlapping square patches of size $P \times P$, leading to:

$$N = \frac{HW}{P^2} \tag{8}$$

  Each patch is flattened and projected into a higher-dimensional embedding space using a learnable matrix E, resulting in:

$$z_{LP} = [x_P^1 E, x_P^2 E, ..., x_P^N E], \quad E \in \mathbb{R}^{(P^2 \cdot C) \times d_{model}} \tag{9}$$

  The classification token is a learnable random vector of size d_{model}

- **Adding the Classification Token & Positional Embeddings**
  A special learnable **classification token** x_{cls} is added to the beginning of the sequence:

$$z_{LP}^{'} = [x_{cls}, x_P^1 E, x_P^2 E, ..., x_P^N E] \tag{10}$$

  Positional embeddings E_{pos} are then incorporated:

$$z_0 = z_{LP}^{'} + E_{pos}, \quad E_{pos} \in \mathbb{R}^{(N+1) \times d_{model}} \tag{11}$$

  These embeddings provide the Transformer with spatial awareness, compensating for its lack of an inherent sense of locality.

- **Transformer Encoding**
  and the resulting z_0 is passed on to the Transformer Encoder. At this point, each encoder block computes the output z_l as follows::

$$\begin{cases} z_l' = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1} \\ z_l = \text{MLP}(\text{LN}(z_l')) + z_l' \end{cases} \tag{12}$$

  where:

  - **MSA** refers to **Multi-Head Self-Attention**.
  - **MLP** is a feedforward neural network.
  - **LN** stands for **Layer Normalization**.

- **Final Prediction Stage**
  Once the final Transformer layer is processed, the **classification token output** is passed to an **MLP head** for the final classification:

$$\text{logits} = \text{MLP}_{\text{head}}(z_L^0) \tag{13}$$

  For object detection tasks, a specialized detection head replaces the classification head.

4.2.3 Self-Attention Mechanism

The self-attention mechanism is a fundamental aspect of transformer-based models. It enables each element in an input sequence to interact with every other element, allowing the model to determine which parts are most relevant for understanding the given data. This mechanism constructs three vectors—query, key, and value—for every input element, which are then used to compute attention scores. These scores help determine the significance of each element in relation to the others. The final output is obtained by computing a weighted sum of the value vectors, where the weights are derived from the relationship between the query and key vectors.

In the field of natural language processing, self-attention plays a crucial role in identifying important words and their relationships within a sentence. For example, in translation tasks, the model must understand how words relate to one another to generate an accurate translation. Self-attention allows the model to focus on relevant words while disregarding less important ones, facilitating high-quality translations.

In the Vision Transformer (ViT) model, self-attention operates using Scaled Dot-Product Attention, which enhances computational efficiency and stability. Mathematically, given an input sequence of n vectors:

$$x = [x^1, x^2, ..., x^n], \quad x \in \mathbb{R}^{n \times d_{model}} \tag{14}$$

three corresponding matrices—Query (Q), Key (K), and Value (V)—are created by multiplying the input with learned parameter matrices:

$$\begin{cases} Q = x \cdot W^Q, & W^Q \in \mathbb{R}^{d_{model} \times d_k}, Q \in \mathbb{R}^{N \times d_k} \\ K = x \cdot W^K, & W^K \in \mathbb{R}^{d_{model} \times d_k}, K \in \mathbb{R}^{N \times d_k} \\ V = x \cdot W^V, & W^V \in \mathbb{R}^{d_{model} \times d_v}, V \in \mathbb{R}^{N \times d_k} \end{cases} \tag{15}$$

The Q and K matrices reside in the same space ($\mathbb{R}^{N \times d_k}$), while the V matrix may exist in a different space ($\mathbb{R}^{N \times d_v}$). The next step is computing the attention scores, represented as matrix S, which captures the relevance of each element relative to a given input:

$$S = Q \cdot K^T, \quad S \in \mathbb{R}^{n \times n} \tag{16}$$

To ensure stable gradients, the score matrix is scaled by $\sqrt{d_k}$, followed by applying a softmax function to normalize the attention weights. These weights are then multiplied by the value matrix V to produce the final output sequence:

$$z = Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V, \quad z \in \mathbb{R}^{n \times d_k} \tag{17}$$

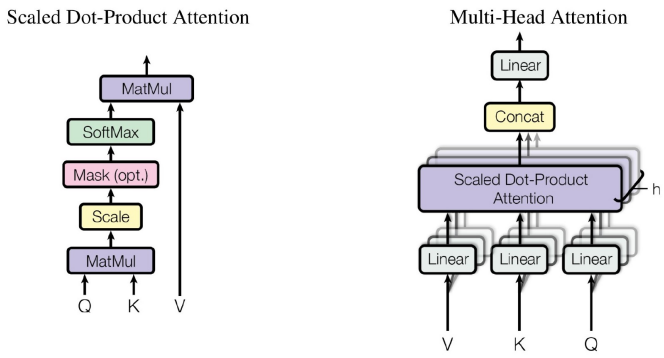

Figure 5. On the left, Scaled Dot-Product Attention. On the right, Multi-Head Attention consists of several attention layers running in parallel. The illustrative picture is taken from [7]

4.2.4 Multi-Head Self-Attention

The Multi-Head Self-Attention (MSA) mechanism is an extension of standard self-attention that enhances the model's ability to process diverse features across multiple perspectives. Instead of relying on a single attention function, MSA leverages multiple attention layers, referred to as attention heads. Each head processes the input independently, capturing distinct relationships between elements in the sequence. This design enables the model to represent information more effectively and learn richer dependencies across different regions of the input.

Each attention head utilizes a separate set of weight matrices, denoted as W_Q, W_K, and W_V, which correspond to queries, keys, and values. These weights are initialized randomly and are refined during training. The outputs of all attention heads are concatenated and then transformed using a learned weight matrix, U_{MSA}, producing the final output representation. This multi-headed approach allows the model to understand multiple aspects of the input data simultaneously, contributing to the overall success of Transformer-based architectures.

$$z = [h^1; h^2; ....; h^h] \cdot U_{MSA}, \quad U_{MSA} \in \mathbb{R}^{h \cdot d_v \times d_{model}} \tag{18}$$

4.2.5 Positional Embeddings

Unlike convolutional neural networks (CNNs), which inherently process spatial information, Vision Transformers (ViTs) treat input images as sequences of patches. However, since Transformers lack an intrinsic understanding of spatial relationships, positional embeddings are incorporated to retain spatial structure. These embeddings encode positional information into the model, helping it recognize how image patches relate to each other. An illustration of positional embeddings in ViT is shown in Figure 6, highlighting

the spatial relationships encoded across image patches.

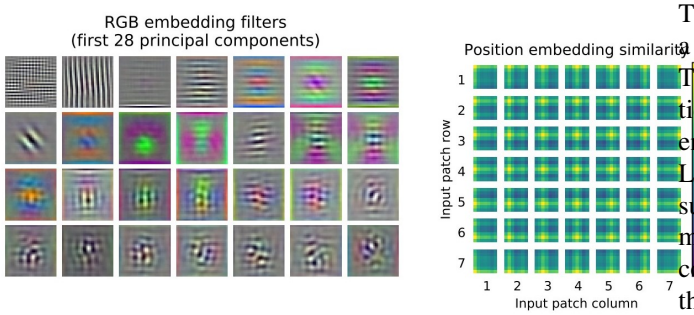

Figure 6. Left: Filters of the initial linear embedding of RGB values of ViT-L/32. Right: Similarity of position embeddings of ViT-L/32. Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches(The image is taken from [6])

Positional embeddings can be introduced in various ways. One common method involves using sinusoidal positional embeddings, which are predefined and added to the input data. Alternatively, ViTs often employ learnable positional embeddings, where the model assigns a unique trainable vector of dimension d_{model} to each patch. These embeddings are learned alongside the model parameters and play a crucial role in maintaining spatial coherence in the representation.

4.3. OWL-VIT

4.3.1 Introduction

Conventional object detectors typically rely on closed-vocabulary classification, where models are trained on a fixed set of object categories and struggle to generalize to novel classes without further fine-tuning. Open-Vocabulary Object Detection (OVOD) addresses this limitation by enabling detection of previously unseen objects through textual descriptions. OWL-ViT (Open-World Learning Vision Transformer) [8] is a state-of-the-art OVOD method that combines contrastive vision-language pre-training with the scalability of Vision Transformers (ViTs). It follows a two-stage pipeline: (1) pre-training an image-text encoder on large-scale image-text pairs using contrastive learning, and (2) fine-tuning lightweight detection heads on standard object detection datasets. OWL-ViT retains separate encoders for images and text, enabling efficient and flexible query processing for open-vocabulary detection.

4.3.2 Model Architecture

OWL-ViT builds on the Vision Transformer (ViT) [6] framework, originally designed for classification tasks, and extends it for open-vocabulary object detection. The key architectural modifications include:Contrastive Pre-Training The image and text encoders are trained contrastively on a large dataset of paired images and text descriptions. This process ensures that the representations of semantically similar image-text pairs are aligned in a shared embedding space, similar to models like CLIP [5], and LiT [9]. This enables the model to learn generalized visual representations without the need for exhaustive human annotations.Transformer-Based Image and Text Encoders.The image encoder is a Vision Transformer (ViT) that processes an input image into a set of feature tokens. The text encoder is a Transformer model that encodes class names or textual descriptions into vector representations.Unlike classical object detectors that use a fixed classification layer, OWL-ViT replaces learned class embeddings with text embeddings obtained from the text encoder. This substitution enables the model to classify objects based on textual queries at inference time. Adaptation for Object Detection The final token pooling layer is removed from the ViT backbone to retain per-token feature representations. A lightweight classification and localization head is attached to each transformer output token. The number of predicted objects is equal to the sequence length of the image encoder (e.g., 576 tokens for ViT-B/32 at 768×768 resolution), which is sufficient given the object instance counts in modern datasets. The model predicts object bounding boxes using a small MLP head attached to the per-token embeddings. it resembles DETR, but is simplified by removing the decoder. The defining feature of OWL-ViT is its ability to perform open-vocabulary detection without requiring explicit retraining on new object categories. This is achieved by replacing traditional object classification layers with text embeddings, allowing detection based on flexible text-based queries.

Text Embeddings as Queries The model classifies detected objects by comparing their embeddings to text-derived embeddings, which are obtained by encoding category names or descriptive text phrases.

Query-Specific Label Space Unlike fixed-label object detectors, OWL-ViT dynamically defines the label space per image based on user-provided queries. This allows the model to:

Detect objects described by arbitrary textual inputs. Support zero-shot detection, where the model generalizes to unseen categories.

OWL-ViT processes each text query separately, rather than merging all queries into a single input sequence. Each query is independently passed through the text encoder, allowing the model to handle a dynamic and flexible label space per image. This approach improves efficiency, as the image encoding does not need to be recomputed for each new query. allowing us to use thousands of queries per im-

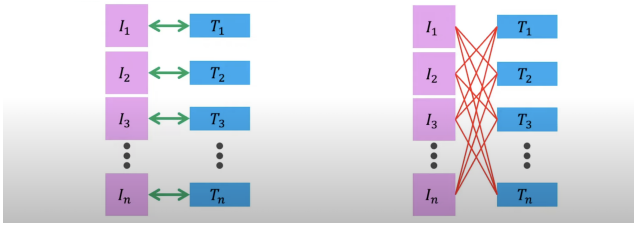

Figure 8. Left: Increasing the cosine similarity of correct pairs. Right: Reducing the cosine similarity of the $n^2 - n$ incorrect pairings.

age.

Figure 7 illustrates the OWL-ViT pipeline, highlighting the contrastive pre-training and adaptation stages for open-vocabulary object detection.

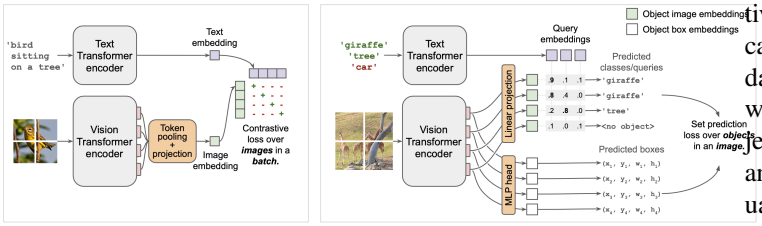

Figure 7. It consists of two main stages. First, we pre-train both an image and text encoder using a contrastive learning strategy with image-text pairs, following a process similar to prior models. Next, we adapt these pre-trained encoders for open-vocabulary object detection by eliminating token pooling and integrating lightweight object classification and localization heads directly into the image encoder's output tokens. Open-vocabulary detection is enabled by embedding query strings through the text encoder, which are then used for classification. The model undergoes fine-tuning on conventional object detection datasets. During inference, it can utilize text-based embeddings for open-vocabulary detection or image-based embeddings for few-shot, image-conditioned detection. The image is taken from [8]

4.3.3 Training Strategy

The training pipeline of OWL-ViT consists of two stages:

- The image and text encoders are trained contrastively using a large-scale image-text dataset. A contrastive loss function ensures that image and text embeddings for matching pairs are drawn closer in representation space, while non-matching pairs are pushed apart.

  The text representation is derived from the end-of-sequence (EOS) token of the text encoder, while the image representation uses multi-head attention pooling (MAP).

- Fine-Tuning for Open-Vocabulary Detection:Fine-tuning pre-trained models for classification is a well-researched area, particularly for large Transformer-based architectures, which require precise regularization and data augmentation techniques to perform effectively. Standard training strategies for classifiers are well-documented in the literature. In this work, we adopt a similar fine-tuning methodology for open-vocabulary object detection.

  The overall training process closely resembles that of closed-vocabulary detectors, with the key difference being that object category names are provided as text-based queries for each image. As a result, instead of using a fixed global label space, the classification head produces logits based on a dynamically defined label space that is specific to each image, determined by the user-specified queries.it uses the bipartite matching loss from DETR [10] but modifies it to suit the requirements open-vocabulary detection. Since exhaustively annotating large detection datasets is impractical, federated annotation [11] is used. In this approach, datasets are constructed as a union of smaller datasets, where each subset is fully annotated for specific object categories. This results in a structured positive and negative set for each category, ensuring that evaluation is focused only on images where category presence is well-defined. Federated datasets help manage the long-tailed distribution of object categories while reducing annotation workload by avoiding unnecessary labeling of frequent categories. Given this dataset structure, focal sigmoid cross-entropy is used instead of softmax cross-entropy to handle class imbalance effectively. Furthermore, positive and negative annotations are used as queries during training, and additional pseudo-negative categories are sampled to ensure at least 50 negative queries per image.

4.4. DETER

DETR [10] 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.

4.4.1 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.

4.4.2 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:

$$f \in \mathbb{R}^{C\times H\times W}, \tag{19}$$

where C is (typically 2048) and

$$H = \frac{H_0}{32}, \quad W = \frac{W_0}{32}. \tag{20}$$

This feature map is then passed into the transformer encoder.

4.4.3 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:

$$z_0 \in \mathbb{R}^{d\times H\times W}. \tag{21}$$

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 $d \times HW$.

Each encoder layer consists of:

- **Multi-head self-attention:** Enables each feature vector to attend to all others, capturing global dependencies.
- **Feed-forward network (FFN):** Applies transformations to enhance feature representation.
- **Positional Encoding:** Since transformers do not have an inherent notion of spatial relationships, fixed positional encodings are added to retain spatial information.

4.4.4 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.

4.4.5 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.

4.4.6 Self-Attention and Encoder-Decoder Attention

Each decoder layer consists of:

- **Multi-head self-attention:** This mechanism allows each object query to attend to all others, helping the model refine predictions by modeling interactions between objects. For instance, if one query attends to an object, another query attending to the same object can be suppressed, reducing duplicate detections.
- **Encoder-decoder attention:** This mechanism enables each object query to attend to relevant image features from the encoder output. Through this attention process, object queries extract information about object locations and categories.

The self-attention mechanism is particularly useful in crowded scenes, where multiple objects may be spatially close to each other. The ability to reason globally helps in assigning each query to a distinct object while avoiding redundant predictions.

4.4.7 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.

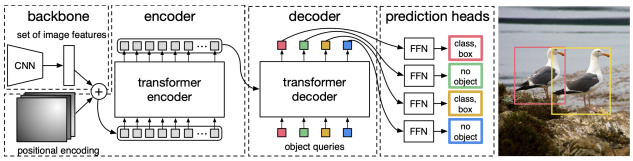

Figure 9. DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "no object" class prediction. Image is taken from [10]

4.4.8 Prediction Feed-Forward Networks (FFNs)

Each object query is processed through a small FFN to generate the final object detection predictions. This module consists of:

- A three-layer perceptron with ReLU activations.
- A linear layer that predicts class probabilities using a softmax function.
- A separate linear layer that predicts bounding box coordinates.

Bounding boxes are predicted in a normalized format relative to the input image dimensions. Since DETR predicts a fixed number of N bounding boxes, an additional label $\emptyset$ is introduced to represent "no object" predictions.

Figure 9 presents an overview of DETR's end-to-end detection pipeline, which combines a CNN and Transformer with bipartite matching during training.

4.4.9 Bipartite Matching with Hungarian Loss

Since DETR predicts a fixed number of objects, a matching strategy is required to associate predictions with ground truth objects. This is done using the Hungarian algorithm, which finds the optimal assignment $\sigma \in S_N$ that minimizes:

$$\hat{\sigma} = \arg\min_{\sigma \in S_N} \sum_{i=1}^{N} L_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \tag{22}$$

where:

- y_i is a ground truth object.
- $\hat{y}_{\sigma(i)}$ is the predicted object assigned to y_i.
- L_{match} is the matching cost.

The matching cost considers classification probability and bounding box similarity.

Once the optimal assignment is found, DETR computes a Hungarian loss:

$$L_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + 1\{c_i \neq \emptyset\} L_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) \right]. \tag{23}$$

The second component of the loss function in DETR is the bounding box loss, wrhich ensures that the predicted bounding boxes are as close as possible to the ground truth boxes. Unlike traditional object detection methods that use anchor boxes and offsets, DETR directly predicts the normalized coordinates of the bounding boxes relative to the input image. To handle variations in object sizes and positions, the bounding box loss is formulated as a combination of two terms:

$$L_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{\text{iou}} L_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L_1} ||b_i - \hat{b}_{\sigma(i)}||_1. \tag{24}$$

where:

- b_i represents the ground truth bounding box for object i.
- $\hat{b}_{\sigma(i)}$ is the predicted bounding box assigned to b_i based on the Hungarian matching.
- $L_{\text{iou}}(b_i, \hat{b}_{\sigma(i)})$ is the generalized Intersection over Union (IoU) loss, which measures how well the predicted and ground truth boxes overlap.
- $||b_i - \hat{b}_{\sigma(i)}||_1$ is the L_1 loss, which ensures precise regression of bounding box coordinates.
- λ_{iou} and λ_{L_1} are hyperparameters that control the relative importance of the two loss terms.

4.5. Fine-Tuning DETR

Object detection is a fundamental task in computer vision, and transformer-based models such as the DEtection TRansformer (DETR) have demonstrated significant improvements over conventional convolutional architectures. DETR, originally designed for detecting objects in the COCO dataset, requires adaptation when applied to specialized domains. In this study, we fine-tuned DETR to detect other objects by modifying its classification layer and optimizing it for our custom dataset.

4.5.1 Adapting DETR to a Custom Dataset

DETR is pre-trained on the COCO [12] dataset, which consists of 91 object classes. Our dataset, however, contains different categories. This discrepancy required modifying the classification head, which originally had 92 output neurons (91 COCO classes plus one background class). We

replaced this layer with a new classifier 8 specifically designed to predict our object classes along with an additional background class.

Since the pre-trained model was optimized for COCO, directly using its classification head would introduce class mismatches and degrade performance. The new classification layer was initialized randomly, ensuring that it was learned from scratch while preserving all other pre-trained weights of the model. This approach allowed us to benefit from DETR's strong feature extraction and object detection capabilities while tailoring it to our specific application.

4.5.2 Training Strategy

To effectively train the modified DETR model, we used a structured fine-tuning approach. The learning rate was carefully adjusted to optimize both the newly introduced classification layer and the pre-trained backbone. A lower learning rate was used for the backbone to prevent excessive modifications to the learned feature representations, while a slightly higher learning rate was applied to the newly initialized classification layer. This ensured stable convergence and prevented catastrophic forgetting of previously learned features.

The training process involved supervised learning with a combination of classification, bounding box regression, and generalized Intersection over Union (IoU) losses. These loss functions allowed the model to accurately classify objects and localize them within an image. Furthermore, data augmentation techniques were applied to improve generalization, including geometric transformations such as rotations, translations, and scale variations.

4.5.3 Evaluation and Performance Assessment

The fine-tuned DETR model was evaluated using standard object detection metrics, including Average Precision (AP) and Average Recall (AR). These metrics provided insights into the model's detection accuracy across different object sizes and confidence thresholds. Comparisons with the original COCO [12]-trained DETR model were conducted to assess the benefits of fine-tuning on a domain-specific dataset.

Through this process, DETR was successfully adapted to recognize other objects with high accuracy. The modifications to the classification head, combined with an optimized training procedure, enabled effective fine-tuning while preserving the advantages of the transformer-based detection framework.

5. 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 [13]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.

5.1. 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:

$$(x, y, w, h, c) \tag{25}$$

  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:

$$c = P(\text{Object}) \times IOU_{\text{pred, truth}} \tag{26}$$

- C class probabilities for each object category.

Thus, the final output tensor size is:

$$S \times S \times (B \times 5 + C) \tag{27}$$

5.1.1 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 $(P(\text{Class}|\text{Object}))$

5.1.2 Pretraining on ImageNet

To build an efficient object detection system, YOLO's convolutional layers are initially **pretrained** on the **ImageNet [14]**. The pretraining phase focuses on feature extraction, where the first **20 convolutional layers** are trained using an average-pooling layer and a fully connected layer [15].

The YOLO network architecture, consisting of convolutional and fully connected layers optimized for real-time detection, is illustrated in Figure 10.

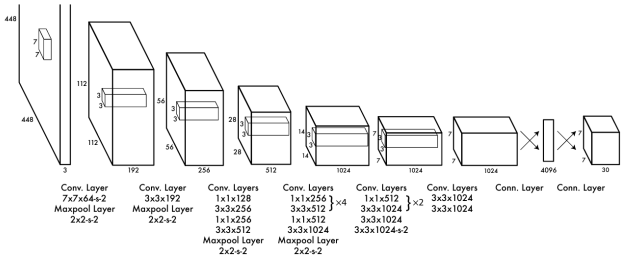

Figure 10. network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 * 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224*224 input image) and then double the resolution for detection.The illustrative picture is taken from [15]

5.1.3 Conversion to Object Detection

After pretraining, the model is converted for object detection by adding:

- **Four additional convolutional layers** for feature refinement.
- **Two fully connected layers** to predict bounding boxes and class probabilities.

This architecture modification enhances fine-grained detection accuracy. Additionally, the input image resolution is increased from 224×224 **to** 448×448 to capture finer details.

5.1.4 Bounding Box Prediction Mechanism

The final layer predicts:

1. **Class probabilities** for each detected object.
2. **Bounding box coordinates** normalized as:
   - The final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell loca tion so they are also bounded between 0 and 1.

A **linear activation function** is used for the final layer, while other layers employ **leaky rectified linear activation**, defined as:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{28}$$

This prevents neuron inactivity and improves gradient flow during training.

5.1.5 Loss Function and Optimization Strategy

YOLO employs a **sum-squared error loss function** to optimize the network. While this approach is computationally efficient, it does not perfectly align with the goal of **maximizing average precision**. One limitation of this loss function is that it **equally weights localization, classification, and confidence errors**, which may not be ideal for object detection.

Additionally, in every image, most grid cells **do not contain any object**. This imbalance leads to strong gradients from background cells that push the confidence scores toward zero, often overpowering the gradients from cells containing objects. This can lead to model instability and cause training to diverge.

To address these issues, YOLO introduces **two key modifications**:

- Increasing the loss from bounding box coordinate predictions by a factor of $\lambda_{\text{coord}} = 5$.
- Decreasing the loss from confidence predictions for boxes that do not contain objects by using $\lambda_{\text{noobj}} = 0.5$.

The complete loss function consists of five main components, which balance localization accuracy, classification accuracy, and confidence scores.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{29}$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{30}$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B} 1_{\text{ij}}^{obj}(C_i - \hat{C}_i)^2 \tag{31}$$

$$+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B} 1_{\text{ij}}^{noobj}(C_i - \hat{C}_i)^2 \tag{32}$$

$$+\sum_{i=0}^{S^2} 1_{\text{i}}^{obj}\sum_{c\in\text{classes}}(p_i(c) - \hat{p}_i(c))^2 \tag{33}$$

where:

- 1_{obj}^{i} denotes if an object appears in cell i, and 1_{obj}^{ij} denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.
- $S \times S$ is the number of grid cells in the image.
- B is the number of bounding boxes per grid cell.
- (x_i, y_i) are the center coordinates of the predicted bounding box.
- (w_i, h_i) are the width and height of the predicted bounding box.
- C_i represents the confidence score predicted for the bounding box.
- $p_i(c)$ is the class probability for object class c.

5.2. Fine-Tuning YOLO Using Ultralytics

Fine-tuning YOLO using the Ultralytics framework involves loading a pre-trained model and training it on our dataset. Different model sizes (e.g., YOLOv8x, YOLOv8s, YOLOv8m, YOLOv8l) can be used depending on the desired trade-off between speed and accuracy. YOLO models predict object classes in the final classification layer. When fine-tuning on a custom dataset with a different number of classes, this layer must be modified. The existing classification head is replaced with a new layer that has outputs matching the number of custom classes. This new layer is randomly initialized and learns to recognize the new object categories during training while the rest of the model retains pretrained weights.

Process:

- The pretrained model's classification head (originally trained on standard datasets) is removed.
- A new classification layer with outputs equal to the custom dataset's classes is added.
- This layer starts with random weights and is trained from scratch.
- The backbone and detection layers retain their pretrained weights, ensuring faster convergence.

6. Experimental Results

In this section, we summarize the results of fine-tuning and evaluating different object detection models on the Roboflow-exported version of the PASCAL VOC 2012 dataset. We focused on three types of architectures: YOLOv8 (single-stage), DETR (transformer-based), and Faster R-CNN (two-stage). For all models, we report key performance metrics including mean Average Precision at IoU 0.5 (mAP@0.5), Precision, and Recall.

6.1. YOLOv8 Results

We started by fine-tuning several variants of YOLOv8, ranging from the smallest and fastest (YOLOv8n) to the most accurate (YOLOv8x). These models were trained using the same data splits and augmentation settings to keep comparisons fair.

Table 1. YOLOv8 Performance Comparison

Model	mAP@0.5	Precision	Recall
YOLOv8n	67	68	68
YOLOv8s	66	68	66
YOLOv8m	69	71	70
YOLOv8l	70	78	79
YOLOv8x	76	79	80

As expected, the larger variants generally performed better in terms of accuracy, though the trade-off is a slower inference time and higher memory usage. YOLOv8m and YOLOv8l ended up being the sweet spot in terms of performance vs. speed for our dataset.

6.2. DETR Results

Next, we trained DETR with two different backbones: ResNet-50 and ResNet-101. DETR takes a very different approach to object detection compared to YOLO and R-CNN — it relies on transformers and doesn't use anchor boxes or non-maximum suppression. This makes the model elegant and end-to-end, but also a bit slower to train.

Table 2. DETR Performance Comparison

Backbone	mAP@0.5	Precision	Recall
ResNet-50	66	68	72
ResNet-101	67	68	73

The ResNet-101 version consistently outperformed the ResNet-50 version, but it also required more compute. DETR handled overlapping objects and cluttered scenes better than YOLO, which was a pleasant surprise. That said, it took much longer to converge, so training time was definitely a consideration.

6.3. Faster R-CNN Results

Faster R-CNN served as a strong baseline. We used the ResNet-50 backbone with a Feature Pyramid Network, which is a fairly standard setup. While it's not designed for speed, its accuracy — especially on smaller and more detailed objects — was solid.

Table 3. Faster R-CNN Performance

Backbone	mAP@0.5	Precision	Recall
ResNet-50 (FPN)	69	74	75

While it didn't outperform the transformer-based DETR or the top YOLO variants in our tests, it was very reliable and handled edge cases well. The downside was the longer inference time and slightly more complex training pipeline.

6.4. What We Learned

- **YOLOv8** gave us the most flexibility across hardware constraints — it was fast, easy to train, and still very accurate, especially in its medium and large variants.
- **DETR** showed the most interesting behavior in crowded scenes or when object boundaries were ambiguous. Its ability to detect cleanly without needing NMS was a real advantage, though the long training time is a drawback.
- **Faster R-CNN** was dependable and often spotted objects that the others missed. But its two-stage nature and slower speed make it more of a benchmark model than a production choice in our setup.

Overall, each model had its strengths. For real-time or edge applications, YOLOv8 is likely the best choice. For projects where training time isn't an issue and clean end-to-end predictions are preferred, DETR is a strong option. And for academic baselines or cases needing detailed localization, Faster R-CNN remains a classic that still performs well.

References

[1] Y. Zhang, J. Lin, X. Han, Y. Hu, Y. Tang, H. Xu, and J. Sun, "Contrastive learning of medical visual representations from paired images and text," *arXiv preprint arXiv:2010.00747*, 2020.

[2] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," *arXiv preprint arXiv:1906.00910*, 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[5] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://arxiv.org/pdf/2010.11929

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[8] A. S. M. N. D. W. A. D. A. M. A. A. M. D. Z. S. X. W. X. Z. T. K. N. H. Matthias Minderer, Alexey Gritsenko, "Simple open-vocabulary object detection with vision transformers," *ECCV*, 2022.

[9] X. Zhai, X. Wang, B. Mustafa, A. Steiner, D. Keysers, A. Kolesnikov, and L. Beyer, "Lit: Zero-shot transfer with locked-image text tuning," *arXiv preprint arXiv:2111.07991*, 2021. [Online]. Available: https://arxiv.org/pdf/2111.07991

[10] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *arXiv preprint arXiv:2005.12872*, 2020.

[11] A. Gupta, P. Dollar, R. Girshick, L. van der Maaten, and K. He, "Lvis: A dataset for large vocabulary instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *ArXiv*, vol. abs/1405.0312, 2014.

[13] R. Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei,

"Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://arxiv.org/pdf/1409.0575

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 779–788. [Online]. Available: https://arxiv.org/abs/1506.02640