The Open University of Israel

Department of Mathematics and Computer Science

End-to-End Object Detection with Neural Networks

Final Paper submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Department of Mathematics and Computer Science

By

Guy Golan

Prepared under the supervision of Dr. Oren Barkan

December 2023

Abstract

End-to-end object detection has emerged as a powerful approach in computer vision, aiming to overcome the limitations of traditional object detection methods that rely on separate stages for detection and post-processing. This paper presents a comprehensive exploration of end-to-end object detection, covering both the underlying deep learning architectures and the state-of-the-art models. We begin with an overview of object detection techniques, including traditional methods like Viola-Jones [1] and Histogram of Oriented Gradients (HOG) [2] and continue with the foundational concepts of convolutional neural networks (CNNs) and their common architectures. We then delve into the advancements in object detection frameworks, including two-stage detectors such as the region-based CNN (RCNN) [3] family and one-stage detectors like You Only Look Once (YOLO) [4]. Finally, we provide an in-depth analysis of the Vision Transformers (ViT) [5] and exploring two approaches for End-to-end detection - the Detection Transformer (DETR) [10], and Pix2Seq [11], which represent the latest advancements in end-to-end object detection. We explore the architectural components, training strategies, and performance evaluations in detail. Through this paper, we aim to provide a comprehensive understanding of end-to-end object detection and shed light on the potential and challenges associated with this exciting field of research.

Table of contents

1. Introduction

Problem statement

Object detection is a fundamental task in computer vision that involves identifying and localizing objects within an image or video. The required result is defined by drawing a bounding box around the object in the image. The bounding box is defined by its four edges, which are parallel to the axes of the image or video (see Figure 1). The goal is to accurately localize the object within the bounding box, such that the box encompasses the object with minimal extra space. This method is called "axis-aligned" because the edges of the bounding box are parallel to the axes of the image, regardless of the orientation or shape of the object itself.

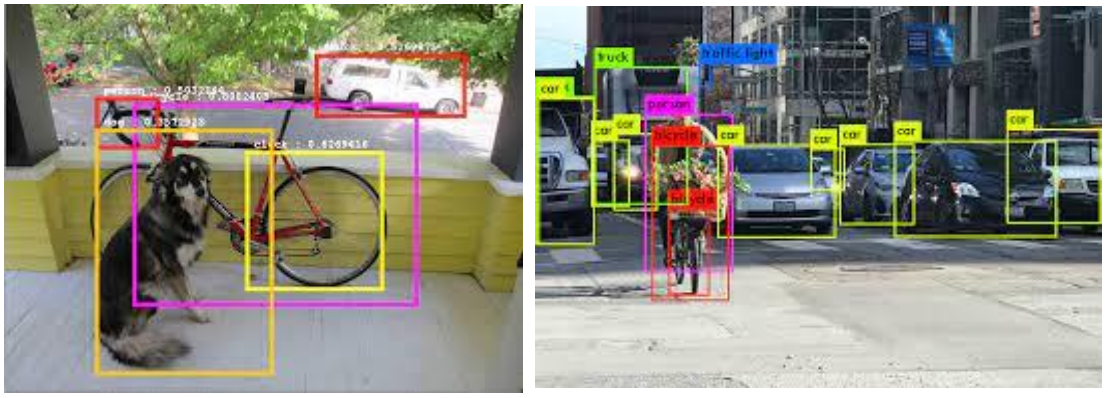

Figure 1: Illustration of the object detection task. Every object detection is defined by a bounding box (rectangle), different classes are colored differently.

Key challenges in object detection

Object detection is a complex problem that involves addressing various challenges. Some of the key challenges are:

- Intra-class variation: Objects within the same class can have significant variations in appearance due to factors such as occlusion, illumination, pose, viewpoint, and other unconstrained external factors. These variations can lead to difficulties in accurately detecting and localizing the objects. Non-rigid deformation, rotation, scaling, and blurriness are some of the other factors that can further complicate the problem.
- Number of categories: Object detection involves identifying objects from a large number of categories, which can make the problem challenging to solve. The high number of object classes also requires a large amount of annotated data for training the detectors, which can be difficult to obtain in practice. One open research question is how to build accurate detectors with fewer training examples.

- Efficiency: With the rise of mobile and edge devices, the need for efficient object detectors has become crucial. Realtime systems like autonomous vehicles must get detection results on a dedicated hardware in a few milliseconds. Current state-of-the-art models require significant computational resources to generate accurate detection results, which is not feasible for many practical applications. Developing more efficient object detectors is a major research challenge in the field of computer vision.

Datasets for object detection

Object detection is a supervised learning task that requires a large amount of annotated data for training and evaluation. Several datasets have been introduced over the years to facilitate research and benchmarking in the field. In this section, we will introduce some of the most used datasets for object detection, such as COCO [9], Pascal VOC [10], and ImageNet [11].

The Common Objects in Context (COCO) [9] dataset is one of the most widely used datasets for object detection. It contains more than 330,000 images with more than 2.5 million object instances labeled across 80 categories. The images in COCO are taken from complex scenes with a diverse range of objects, making it a challenging dataset for object detection. The dataset is split into three sets: train, validation, and test sets. The train set contains 118,287 images, the validation set contains 5,000 images, and the test set contains 40,000 images. The annotations for COCO include bounding boxes and segmentation masks for each object instance, as well as labels for each object category.

The Pascal VOC (Visual Object Classes) [10] dataset is another popular dataset for object detection. It was introduced in 2005 and has been used as a benchmark dataset for object detection ever since. The dataset contains images from 20 different object categories, including animals, vehicles, and household objects. The dataset is split into three sets: train, validation, and test sets. The train set contains around 5,000 images, the validation set contains around 5,000 images, and the test set contains around 5,000 images. The annotations for Pascal VOC include bounding boxes for each object instance, as well as labels for each object category.

ImageNet [11] is a large-scale image database with more than 14 million labeled images across 20,000 categories. It was introduced in 2009 and has been used for various computer vision tasks, including object detection. However, it is not specifically designed for object detection, and the annotations for object detection are only available for a subset of the images. The ImageNet detection challenge is a subtask of the

ImageNet challenge, where the goal is to detect objects in a set of 200 categories. The dataset is split into train and validation sets, with 1.2 million and 50,000 images, respectively.

Apart from these datasets, there are other datasets that have been introduced for specific object detection tasks. For example, the KITTI dataset contains images and annotations for detecting objects in autonomous driving scenarios, including cars, pedestrians, and cyclists. The dataset includes around 7,500 images for training and testing, with annotations for each object instance.

Evaluating object detectors

Object detectors use multiple criteria to measure the performance. The most common evaluation metric is mean Average Precision (mAP).
First, we define the Intersection over Union (IoU), which is the ratio of the area of overlap and the area of union between the ground truth and the predicted bounding box (see Figure 2). A threshold is set to determine if the detection is correct (usually 0.5 or higher). If the IoU is more than the threshold, it is classified as True Positive while an IoU below it is classified as False Positive. If the model fails to detect an object present in the ground truth, it is termed as False Negative.

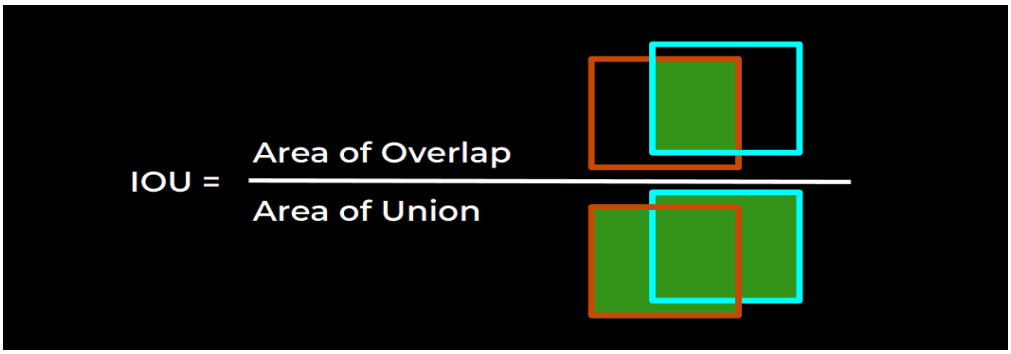

Figure2: Intersection over union definition and illustration.

Precision measures the percentage of correct predictions while the recall measures the correct predictions with respect to the ground truth.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{All\ Observations}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{All\ Ground\ Truth}$$

Based on the above equation, average precision (AP) is computed separately for each class. To compare performance between the detectors, the mean of average precision of all classes, called mean average precision (mAP) is used, which acts as a single metric for final evaluation.

Another commonly used evaluation metric is the F1-score, which is the harmonic mean of precision and recall. The F1-score provides a single number that balances the trade-off between precision and recall.

2. Foundational Approaches to Object Detection

Classical methods for object detection

HOG

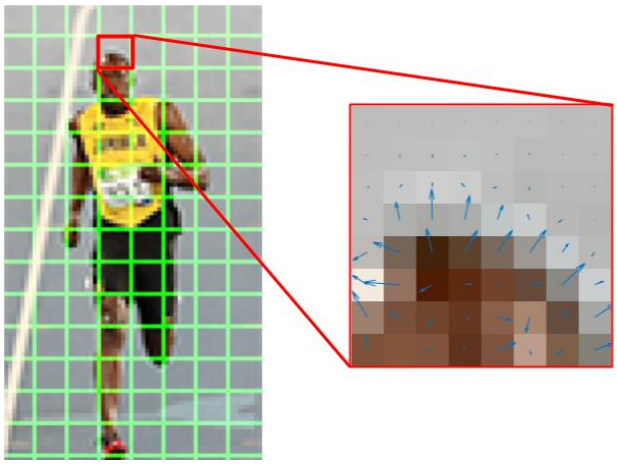

Figure 3: HOG detector. The RGB patch and gradients represented using arrows.

Histogram of Oriented Gradients (HOG) [2] is a feature descriptor that is used for object detection. HOG extracts the shape and texture features of an object by computing the gradients of an image. It works by dividing the image into small cells and computing the histogram of gradient orientations in each cell. The HOG descriptor then concatenates the histograms of the cells in a block-wise fashion to form a feature vector that can be used for object detection.

HOG was first introduced by Navneet Dalal and Bill Triggs in 2005 as a method to detect pedestrians in images. It has since been used for many other object detection tasks such as face detection, vehicle detection, and animal detection.
The HOG feature extraction process involves several steps. First, the image is converted to grayscale and smoothed using a Gaussian filter to reduce noise. Then, the gradients of the image are computed using the Sobel operator, which is a simple edge detection filter. The gradient magnitudes and orientations are then computed for each pixel in the image (Figure 3).

Next, the image is divided into small cells, typically 8x8 pixels, and the gradient orientations within each cell are quantized into several bins. The most common binning scheme uses 9 bins covering a range of 0 to 180 degrees. The gradient magnitudes are weighted by their corresponding gradient orientations and accumulated into the histogram bins.

To increase robustness to lighting and contrast changes, the HOG descriptor also includes normalization steps. First, the histograms within each cell are normalized using a block-wise normalization scheme. Each block contains a fixed number of cells and overlaps with adjacent blocks. The histograms within each block are concatenated into a single feature vector and then normalized to have unit L2 norm. This normalization step ensures that the descriptor is invariant to small changes in lighting and contrast.
Finally, the HOG descriptor is used as input to a machine learning algorithm such as a support vector machine (SVM) for object detection. The SVM learns a decision boundary between positive and negative examples in the feature space, which can be used to classify new images.

While HOG was a significant improvement over earlier feature extraction methods such as SIFT and SURF, it has since been largely supplanted by convolutional neural networks (CNNs) for object detection.

Viola-Jones

The Viola-Jones algorithm [1] is a classic approach for object detection that uses Haar-like features and a machine learning-based approach to detect objects in an image. The algorithm works by scanning an image with a sliding window, and for each window, computing a set of features that describe the image content. These features are computed using Haar-like features, which are rectangular features that compute the difference between the sum of the pixel intensities in two or more adjacent rectangles.

The algorithm then applies a simple machine learning algorithm, called AdaBoost, to select a small number of the most relevant features and use them to classify the object in the image. AdaBoost is a boosting algorithm that iteratively selects the best features and trains a weak classifier on these features. The final classifier is a combination of these weak classifiers.

Viola-Jones is known for its fast speed and good accuracy on face detection tasks. However, it has some limitations, such as being sensitive to illumination changes and being less effective on complex and cluttered scenes. The algorithm also requires careful tuning of its parameters to achieve good performance.
Overall, Viola-Jones was a significant contribution to the field of object detection, paving the way for modern approaches based on deep learning.

Convolutional neural networks

Introduction to CNNs

Convolutional Neural Networks (CNNs) are a type of neural network that has been widely used in computer vision tasks, including object detection. CNNs are inspired by the visual cortex of animals, which is known to be highly efficient in detecting visual patterns. The basic idea of a CNN is to learn a set of filters that can extract meaningful features from the input image and use these features to make predictions.

The input to a CNN is typically an image, represented as a 3D tensor of pixel values (height x width x channels). The first layer of a CNN is a convolutional layer, which applies a set of learnable filters to the input image. Each filter is a small matrix of weights that slides over the input image, performing a dot product between the filter and the image pixels at each location. This operation produces a feature map, which highlights the presence of a certain pattern in the input image.

The convolutional layer is followed by a non-linear activation function, such as the Rectified Linear Unit (ReLU). The ReLU function sets all negative values in the feature map to zero, while leaving positive values unchanged. This operation introduces non-linearity to the network, allowing it to learn more complex patterns.

After the activation function, the feature map is usually downsampled using a pooling layer, which reduces the spatial dimensions of the feature map while preserving the most important information. The most common pooling operations are max pooling, which takes the maximum value in each pooling region, and stride convolution, which applies in the convolution operation within skips.

The process of convolution, activation, and pooling is repeated multiple times, forming a deep network of feature extractors. For image classification task, the final layers of the network are typically fully connected layers, which take the output of the convolutional layers and use it to make classification prediction. In the case of object detection, the fully connected layers are usually connected to a set of output nodes that correspond to the possible object classes, and the network outputs the class probabilities and bounding box coordinates for each object in the image.

One of the main advantages of CNNs is their ability to learn hierarchical representations of the input image, starting from low-level features such as edges and corners, and gradually building up to high-level concepts such as object parts and textures. This

allows the network to capture both local and global context, making it robust to variations in object appearance and occlusions.

CNNs have achieved remarkable success in object detection, outperforming traditional hand-crafted feature extractors such as HOG and Viola-Jones. CNN-based object detectors have been used in a variety of applications, including autonomous driving, surveillance, and robotics.

In the next section, we will dive deeper into the components of a CNN, including convolutional layers, pooling layers, activation functions, and popular CNN architectures.

Basic components of CNNs

Convolutional layers

Convolutional layers are the building blocks of a CNN. They are designed to detect local patterns or features in the input data by performing a mathematical operation known as convolution. Convolutional layers consist of a set of learnable filters or kernels, which are small matrices that slide over the input data and perform element-wise multiplication with the local region of the input that they are currently aligned with. The result of the convolution operation is a single value that represents the degree of similarity between the filter and the input region (see Figure 4). This process is repeated for every possible local region in the input data, resulting in a 2D output feature map.

Convolutional layers have several important properties that make them well-suited for image classification and object detection tasks. First, they can capture spatial relationships between adjacent pixels, which is important for identifying patterns in images. Second, they can learn local, translation-invariant features that can be reused across the entire image. Finally, by stacking multiple convolutional layers together, CNNs are able to learn increasingly complex features and patterns.

One important consideration when designing convolutional layers is the size of the filters. Smaller filters are able to capture more fine-grained details in the input data but may be prone to overfitting. Larger filters can capture more global patterns in the data but may not be as effective at capturing local features.

Another important parameter is the stride, which determines the step size at which the filters move over the input data. A larger stride results in a smaller output feature map and a reduction in computational complexity but may also result in a loss of information.

Overall, convolutional layers are a powerful tool for learning feature representations in image data and are a key component of modern CNN architectures.

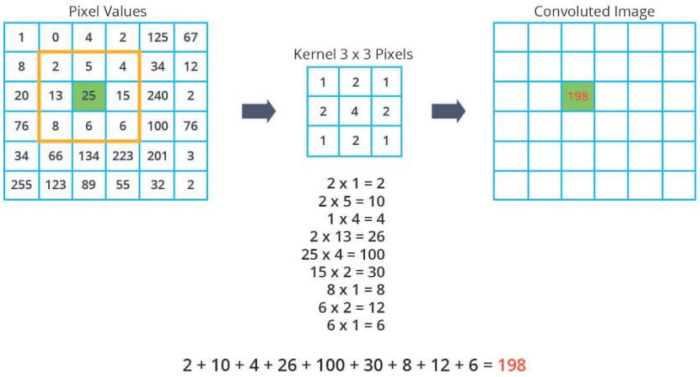

Figure 4: A 3X3 convolution operator illustrated.

Down-sampling methods: Pooling and Stride

There are two common approaches for down-sampling feature maps in convolutional neural networks: pooling and stride.

Pooling (see Figure 5) involves dividing the feature map into non-overlapping regions and computing a summary statistic for each region, such as the maximum value (max pooling) or the average value (average pooling). Pooling reduces the spatial resolution of the feature map, while keeping the number of channels constant. One advantage of pooling is that it introduces a degree of translation invariance by effectively summarizing the presence of a feature in each region rather than its exact location. However, pooling can also result in loss of information, especially for small objects or fine-grained features. Moreover, pooling is typically applied independently to each

channel, which can lead to poor performance when different channels contain complementary information.

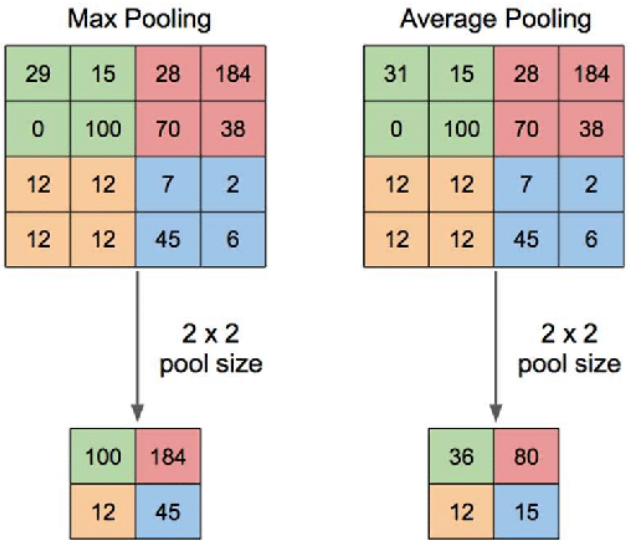

Figure 5: The pooling operation. In max pooling (left) the highest value is take while in average pooling (right) the average of the pixels is calculated.

Stride involves convolving the input feature map with a filter with a larger stride (i.e., the step size between adjacent filter positions) than 1. This results in a down-sampled output feature map with a smaller spatial resolution and a reduced number of channels (See Figure 6). Stride preserves more spatial information than pooling since each output pixel corresponds to a convolutional kernel centered at a particular location in the input feature map. Stride also allows for more flexible control over the down-sampling factor and can be combined with other convolutional operations, such as dilated convolution, to capture multi-scale information.

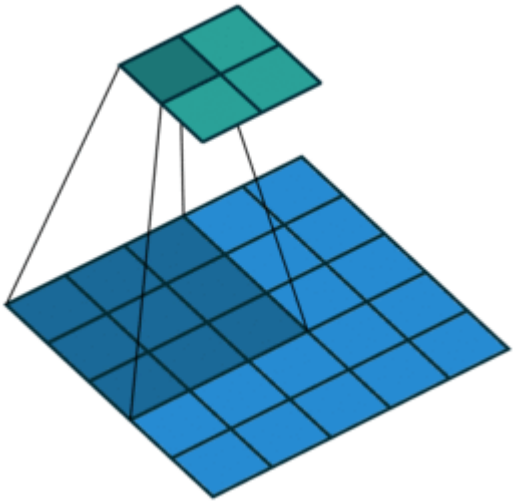

Figure 6: Stride illustrated. Convolution is applied with skipping, which results a downscaled image.

Activation functions

Activation functions are a critical component of convolutional neural networks (CNNs). They introduce non-linearity, which is necessary to model complex non-linear relationships between the inputs and outputs of a network. Without activation functions, the network would be limited to linear transformations, and hence would be less powerful in representing complex patterns and relationships.

There are several activation functions that can be used in CNNs. Some of the earlier ones include the sigmoid and tanh functions, which were commonly used in the past but have fallen out of favor in recent years due to some of their limitations. These functions have a range between `0` and `1` or `-1` and `1`, respectively, which can cause gradients to vanish during backpropagation. This means that the network can become difficult to train as the updates to the weights become smaller and smaller.

In contrast, the rectified linear unit (ReLU) activation function has become the most popular choice in recent years due to its simplicity and effectiveness. The ReLU function outputs the input value if it is positive, and `0` otherwise (see Figure 7). This function is much faster to compute than sigmoid and tanh, and has a much larger range of positive values, making it less prone to gradient vanishing. Additionally, the sparsity induced by ReLU has been shown to help with regularization, reducing overfitting and improving generalization.

Despite its popularity, ReLU does have some limitations. The most prominent issue is the problem of "dead" neurons, where the output of a neuron is always zero due to a negative bias. This can cause a loss of representational power and reduce the effectiveness of the network. To address this issue, several variants of ReLU have been proposed, such as leaky ReLU, which adds a small positive slope to the negative region, and exponential linear units (ELUs), which provide a smooth transition between the negative and positive regions.

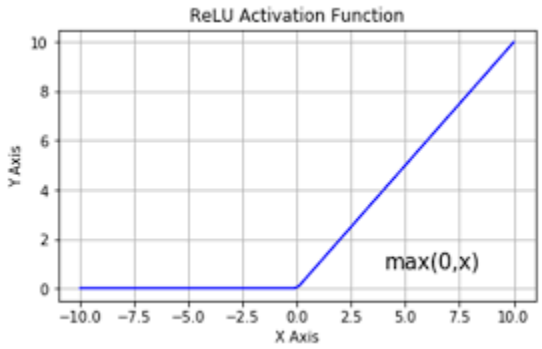

Figure 7: The ReLU function. Values bellow zero are mapped to zero, while values greater than zero aren't changed.

Fully connected layers

Fully connected layers (also known as dense layers) are used in neural networks to connect every neuron in one layer to every neuron in the next layer (see Figure 8). In CNNs, fully connected layers are typically used at the end of the network to produce the final output, such as the class probabilities in a classification task.

Fully connected layers are similar to the traditional Convolutional neural networks in that every neuron in one layer is connected to every neuron in the next layer. However, they lack the spatial invariance property that convolutional layers provide, since the weights are not shared across spatial positions. This means that fully connected layers require a large number of parameters, which can lead to overfitting and make the network difficult to train.

To address this issue, many CNN architectures use a combination of convolutional and fully connected layers, with the fully connected layers placed at the end of the network. The output of the last convolutional layer is typically flattened and fed into the fully connected layers, which produce the final output.

While fully connected layers have some disadvantages, they can still be useful in certain situations, such as when dealing with small input sizes or when the spatial structure of

the data is not important. However, in most cases, using a combination of convolutional and fully connected layers is the preferred approach.

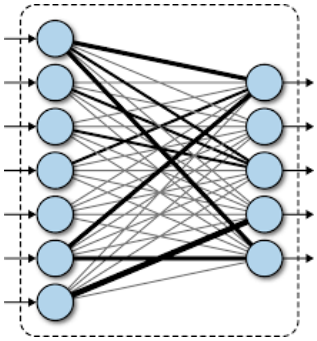

Figure 8: Fully connected layer. Every neuron is connected to the next layer by a dedicated learned weight.

Common CNN architectures

Convolutional Neural Networks (CNNs) have been widely used for image classification tasks, with many notable architectures developed over the years. While classification and object detection are different tasks, the underlying CNN architecture is similar for both. Understanding the evolution of CNNs for classification can provide insights into the development of CNNs for object detection, which we will discuss in the next section. In this section, we'll explore some of the most prominent CNN architectures designed for classification tasks.

LeNet-5

LeNet-5 [13], developed by Yann LeCun et al. in 1998, is one of the earliest CNN architectures for image classification. It consists of two convolutional layers followed by two fully connected layers. LeNet-5 was designed to classify handwritten digits from the MNIST dataset, achieving an error rate of 0.95% at the time.

AlexNet

AlexNet [14], developed by Alex Krizhevsky et al. in 2012, is a more complex CNN architecture that consists of eight layers, including five convolutional layers and three fully connected layers (see Figure 9). It was the first CNN architecture to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, achieving an

error rate of 15.3%, which was a significant improvement over the previous best result of 26.2%.

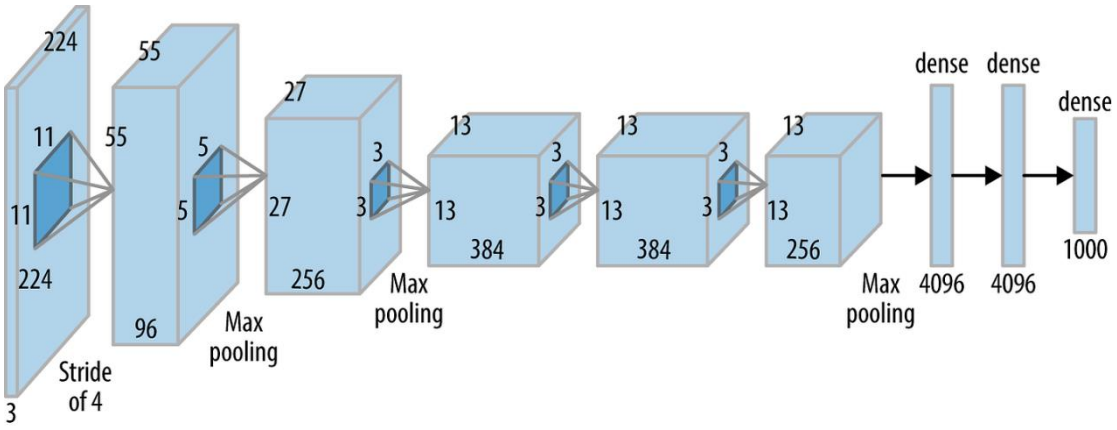

Figure 9: AlexNet architecture. 5 layers of convolutions are followed by three dense layers.

VGGNet

VGGNet [15], developed by Karen Simonyan and Andrew Zisserman in 2014, is a deeper and more complex CNN architecture than AlexNet. It consists of up to 19 layers, with most of them being 3x3 convolutional layers (See Figure 10). VGGNet achieved impressive performance on the ILSVRC, with a top-5 error rate of 7.3%.

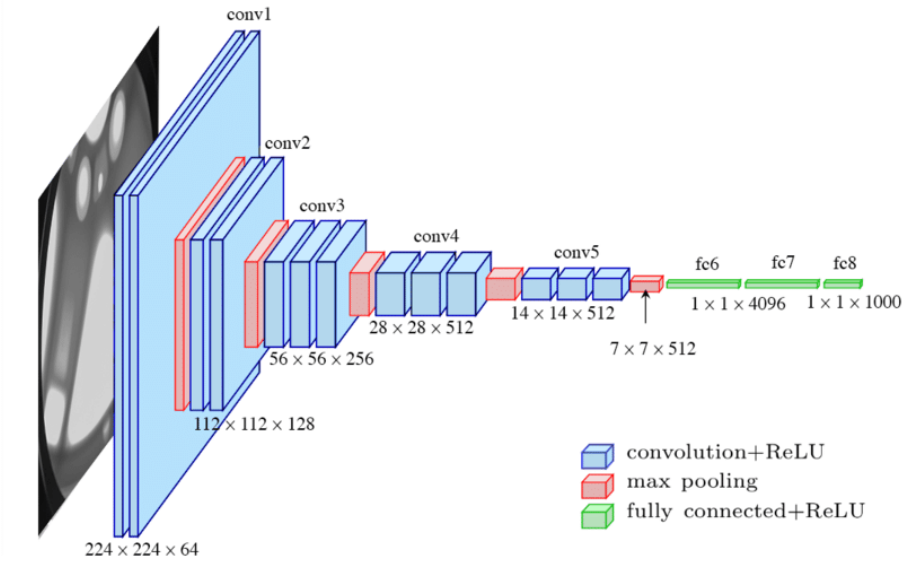

Figure 10: VGG architecture. Deeper and more complex than older architectures.

GoogLeNet/Inception

GoogLeNet [16], developed by Christian Szegedy et al. at Google in 2014, introduced the Inception module, which allowed for more efficient and effective use of convolutional

layers (See Figure 11). The GoogLeNet architecture consists of 22 layers and achieved a top-5 error rate of 6.7% on the ILSVRC.

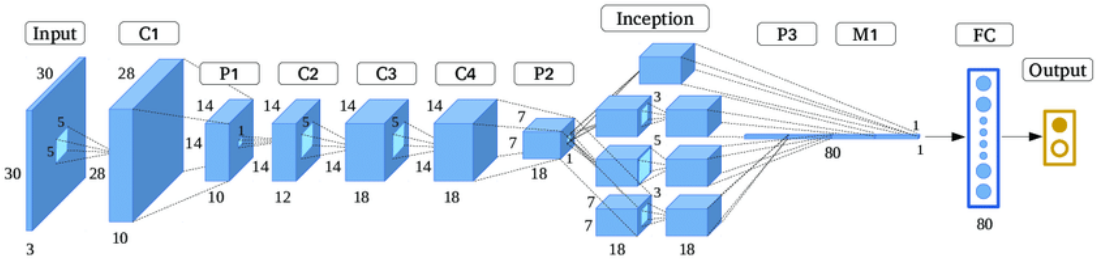

Figure 11: GoogleNet architecture. Improving the SOTA performence a bit more.

ResNet

ResNet [17], developed by Kaiming He et al. at Microsoft Research in 2015, introduced the concept of residual connections (see Figure 12), allowing for deeper CNN architectures to be trained more effectively and address the vanishing gradient problem. ResNet consists of up to 152 layers and achieved a top-5 error rate of 3.6% on the ILSVRC, which was a significant improvement over previous architectures.

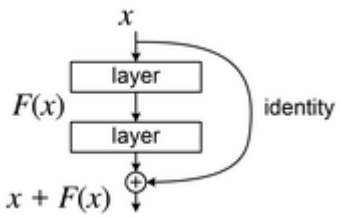

Figure 12: A residual block. input of first layer is connected also to a deeper layer.

Summary

These architectures are all designed for image classification tasks, and each one has its unique strengths and weaknesses. While LeNet-5 is a simple architecture that was the first to demonstrate the effectiveness of CNNs, ResNet is currently one of the most powerful architectures due to its ability to train very deep networks.

In the next section, we'll discuss CNN architectures designed specifically for object detection tasks, which build upon and extend the foundational principles and techniques we discussed earlier.

State-of-the-art object detection frameworks

In the previous section, we explored the basic components of convolutional neural networks (CNNs) and discussed some of the most common CNN architectures, such as AlexNet, VGG, and ResNet. While these models have achieved significant success in image classification tasks, they are not specifically designed for object detection. In recent years, state-of-the-art object detection frameworks have emerged that build upon the foundations laid by these CNN architectures.

In this section, we will explore some of the most advanced object detection frameworks available today. These models can detect objects in complex images with remarkable accuracy and speed. We will start by discussing two-stage object detection frameworks, which include region-based CNNs and their variants such as Fast R-CNN and Faster R-CNN. We will then explore one-stage detectors, including You Only Look Once (YOLO) and its various iterations. Finally, we will discuss some recent advancements in object detection, including detectors based on feature pyramid networks (FPN).

Two-stage Detectors

Two-stage detectors, also known as region-based detectors, are generally follow a two-step procedure: first, they generate region proposals, and then they classify the proposals as either object or background.

Region-based CNN (RCNN) [3] was the first two-stage object detection framework that introduced the concept of region proposals. The RCNN model proposes regions of interest (ROIs) using a selective search algorithm, which generates a large number of candidate regions from an image. The selective search algorithm is used to propose potential object locations and generate region proposals. These proposals are then warped to a fixed size and passed through a convolutional neural network (CNN) to extract features. The extracted features are then used for classification and bounding box regression (see Figure 13). Despite its success, RCNN is computationally expensive, with a slow training and testing time, making it impractical for real-time applications.

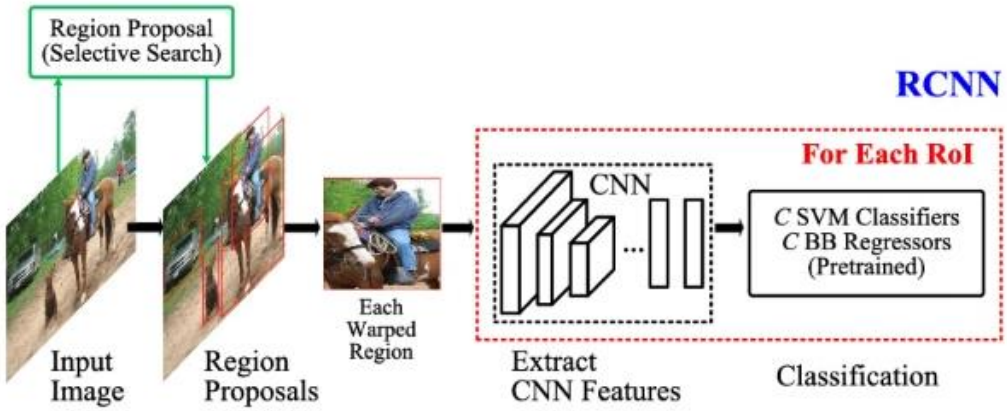

Figure 13: RCNN stages. First, we get the proposals by selective search, then, for each patch - Extract CNN features, and finally classify.

Fast R-CNN was introduced as an improvement over RCNN, where the region proposals are generated from feature maps instead of the input image, making it faster than RCNN (see Figure 14). Fast R-CNN first passes the entire image through a CNN, which generates feature maps. Then, region proposals are generated using selective search from these feature maps. Finally, the region proposals are pooled into a fixed-size feature map and passed through a fully connected layer for classification and bounding box regression. Experiment results showed that Fast R-CNN had 66.9% mAP while R-CNN of 66.0% on PASCAL VOC 2007 dataset and inference time is about 200 times faster.

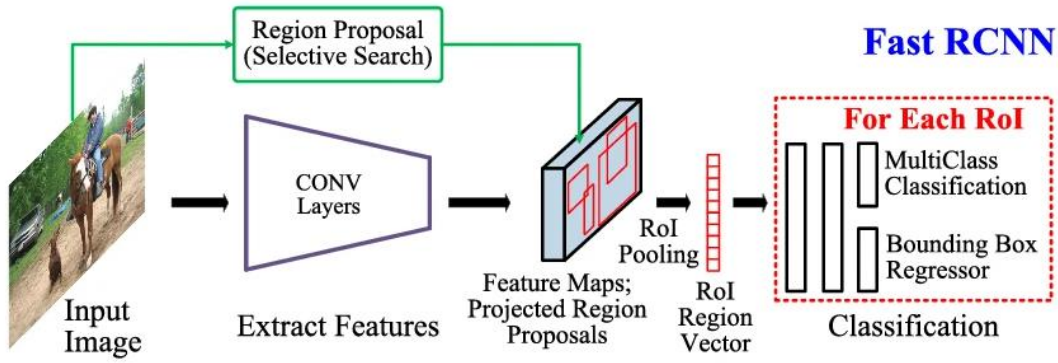

Figure 14: Fast RCNN. The main difference from RCNN is the proposals are generated from a downscaled feature map.

Faster R-CNN is an extension of Fast R-CNN that replaces the selective search algorithm with a region proposal network (RPN) to generate region proposals. RPN is a small CNN that takes the feature maps generated from the input image as input and outputs region proposals. The RPN generates region proposals using anchor boxes,

which are pre-defined bounding boxes with different scales and aspect ratios. Faster R-CNN achieved state-of-the-art results with improved speed over RCNN and Fast R-CNN (mAP of 69.9%, nearly 10 times lower inference time).

One-stage Detectors

You Only Look Once (YOLO) is a family of object detection models that belong to the one-stage detectors category. The first YOLO model was introduced in 2016 by Joseph Redmon et al. Since then, several variants of YOLO have been proposed, including YOLOv2, YOLOv3, and YOLOv4. In this section, we will discuss the key concepts and improvements introduced in each of these models.

YOLOv1 was the first model in the YOLO family, and it proposed a new approach to object detection. Unlike traditional object detection methods, YOLOv1 used a single convolutional neural network to predict both the bounding boxes and the class probabilities for each object in the image. This made the model faster and more efficient than the two-stage detectors like RCNN.

The architecture of YOLOv1 consists of a single convolutional neural network that takes the entire image as input and divides it into a grid of cells (see Figure 15). Each cell in the grid predicts a fixed number of bounding boxes and their corresponding class probabilities. The bounding boxes are represented by their x and y coordinates, height, and width, and they are normalized by the image size. The class probabilities are predicted using SoftMax, and they represent the probability of each object class for each bounding box.

One of the key features of YOLOv1 was its ability to detect objects at different scales and aspect ratios. The model achieved this by using anchor boxes, which are predefined bounding boxes with different sizes and aspect ratios. The model learns to adjust the anchor boxes to better fit the objects in the image.

While YOLOv1 was a significant improvement in terms of speed and efficiency, it had some limitations. One of the main limitations was its accuracy, especially for small objects. YOLOv1 also struggled with detecting overlapping objects and objects with complex shapes.

YOLOv2 was introduced in 2017 and addressed some of the limitations of YOLOv1. The main improvement introduced in YOLOv2 was the use of a new architecture called Darknet-19. Darknet-19 was designed specifically for object detection and achieved better accuracy than the previous architecture used in YOLOv1.

Another improvement introduced in YOLOv2 was the use of batch normalization. Batch normalization is a technique that helps the model converge faster by normalizing the input to each layer. YOLOv2 also used a technique called anchor box clustering to better predict the anchor boxes for each object.

YOLOv2 also introduced a new training technique called multi-scale training. This technique involved training the model on images at different scales and then combining the predictions to improve accuracy. YOLOv2 also used a new loss function that combined localization and classification losses, which improved the accuracy of the model.

YOLOv3 was released in 2018 and introduced significant improvements over its predecessors. It introduced several modifications to the original YOLO architecture that improved its accuracy and speed, including feature pyramid networks (FPNs) and shortcut connections.

FPNs allow for multi-scale feature extraction, which means that objects of different sizes can be detected more effectively. FPNs are essentially a set of convolutional layers that extract features at different scales, and then combine them to produce a set of feature maps that are used to predict the final bounding boxes and class labels.

Shortcut connections, also known as skip connections, allow for the direct flow of information between layers that are far apart in the network architecture. This can help to mitigate the problem of information loss that occurs in very deep neural networks.

YOLOv3 was further improved in 2020 with the release of YOLOv4. This version introduced a number of additional improvements, including a more complex backbone network, advanced data augmentation techniques, and a more accurate loss function. The backbone network used in YOLOv4 is the CSPDarknet-53, which is a modified version of the Darknet-53 backbone used in YOLOv3. The CSPDarknet-53 architecture reduces the computational cost of the network by using cross-stage partial connections, which connect the early and late stages of the network.

In addition to the improvements in the backbone network, YOLOv4 also introduced a number of data augmentation techniques that help to improve the accuracy of the model. These techniques include mosaic data augmentation, which involves combining multiple images into a single training image, and cutout data augmentation, which randomly removes a portion of the image during training.

Finally, YOLOv4 introduced a new loss function called the focal loss, which is designed to address the problem of class imbalance in object detection datasets. The focal loss assigns higher weights to hard examples (i.e., examples that are misclassified by the network with high confidence) and lower weights to easy examples (i.e., examples that are correctly classified by the network with high confidence).

Overall, YOLO and its variants have made significant contributions to the field of object detection, by introducing highly efficient and accurate object detection frameworks that can operate in real-time.
These models have enabled a wide range of applications, including autonomous driving, robotics, and surveillance systems.

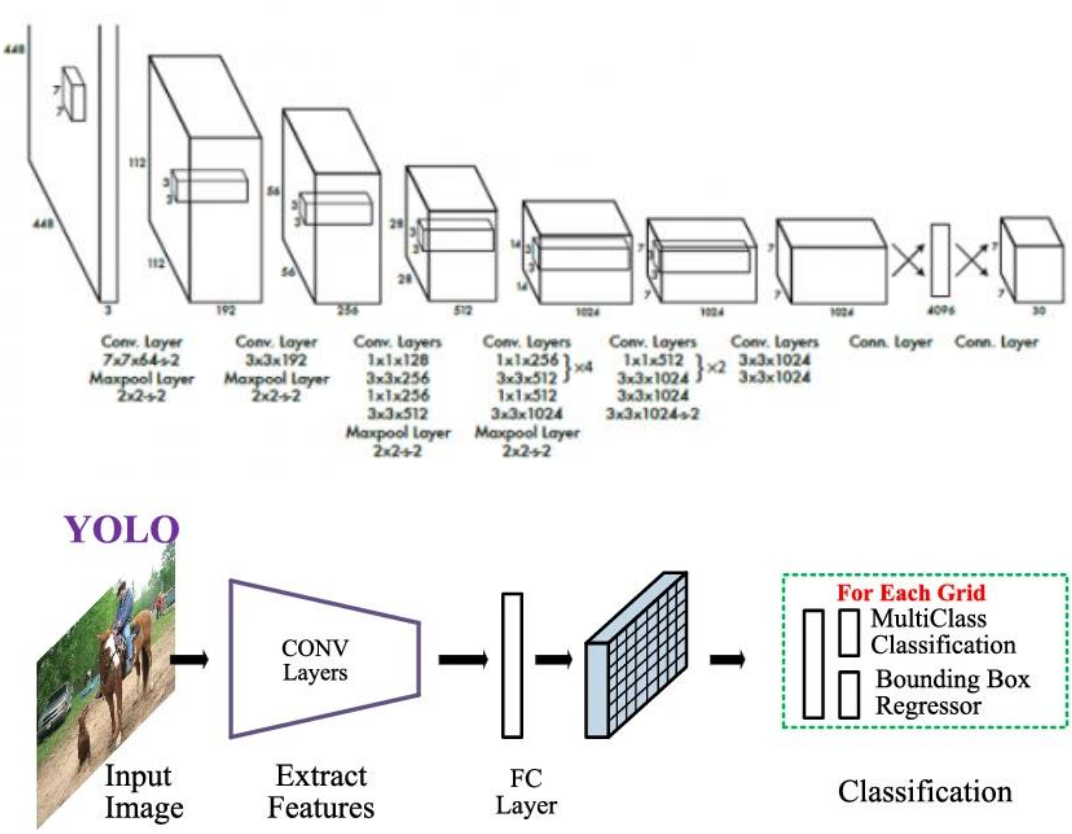

Figure 15: YOLO architecture in details and in high level.

Post processing

It's important to note that the object detection frameworks we've discussed so far often require post-processing techniques to refine their output. One common technique is non-maximum suppression (NMS) [12], which is used to remove duplicate detections and improve the accuracy of the final output.

During inference, the detection framework generates multiple candidates bounding boxes for each object in the image. However, since the model is not perfect, some of these boxes may be inaccurate and overlapping. NMS is applied to these candidate boxes to remove duplicates and retain only the most confident detections.

The NMS algorithm works by first sorting the candidate boxes based on their confidence scores (i.e., the probability that the box contains an object). It then starts with the box with the highest confidence score and iterates over the remaining boxes to check if they overlap significantly with the current box. If the overlap is above a certain threshold (e.g., 0.5), the box is discarded, otherwise, it is kept as a valid detection (see Figure 16).
The gap between the loss function used to optimize the network output and the evaluation metrics used to measure the performance of object detection models is one of the reasons why post-processing methods like NMS can introduce some inaccuracy. While the loss function is designed to directly optimize the network output, the evaluation metrics are calculated after the post-processing step. Therefore, post-processing methods are often heuristic and hand-crafted, and may not perfectly align with the ultimate goal of the network. This misalignment can lead to a gap between the optimized network output and the evaluation metrics, resulting in some inaccuracies in the final predictions.

Figure 16: non maximum suppression (NMS). Overlapping boxes are being suppresssed so only one box (with the highest confidence) survives

Transformers

Introduction to transformers

Transformers have emerged as a powerful class of models in various domains, including natural language processing (NLP) and computer vision. Originally introduced for NLP tasks, transformers have shown great potential in solving complex problems that require modeling long-range dependencies and capturing global context. In recent years, they have gained significant attention and have been successfully applied to computer vision tasks, including object detection.

At the heart of transformers lies their ability to leverage self-attention mechanisms, which enable the models to attend to different parts of the input sequence and capture meaningful relationships. This attention mechanism allows the model to assign varying weights to different positions, emphasizing more important information and effectively modeling interactions between elements. This capability is particularly advantageous in tasks such as object detection, where capturing contextual information and understanding object relationships are crucial.

One of the key benefits of transformers in object detection is their ability to process the entire image in parallel, unlike traditional convolutional neural networks (CNNs) that rely on sequential processing and hierarchical feature extraction. By considering the global context of the image, transformers can potentially capture long-range dependencies and learn more comprehensive representations. This global perspective is especially valuable for object detection, where objects may appear at various scales and locations within the image.

Another advantage of transformers is their capacity to capture rich semantic information. While CNNs excel at capturing low-level visual features, transformers offer a more abstract representation by modeling higher-level semantics. This semantic understanding allows transformers to reason about object classes, relationships, and attributes, leading to more accurate object detection.

In the context of object detection, transformers have been successfully applied as backbone architectures or as complete end-to-end solutions. Vision transformers (ViTs) have gained prominence as a specific class of transformers designed for image-based tasks. ViTs typically divide the image into fixed-size patches and flatten them into a sequence of tokens, which are then processed by the transformer layers. This approach allows transformers to handle images of arbitrary sizes and facilitates parallel processing.

The next sections will delve into the key components of transformers, including the self-attention mechanism, multi-head attention, and the architecture of transformer encoders. The Transformer block is summarized and illustrated in Figure 17.

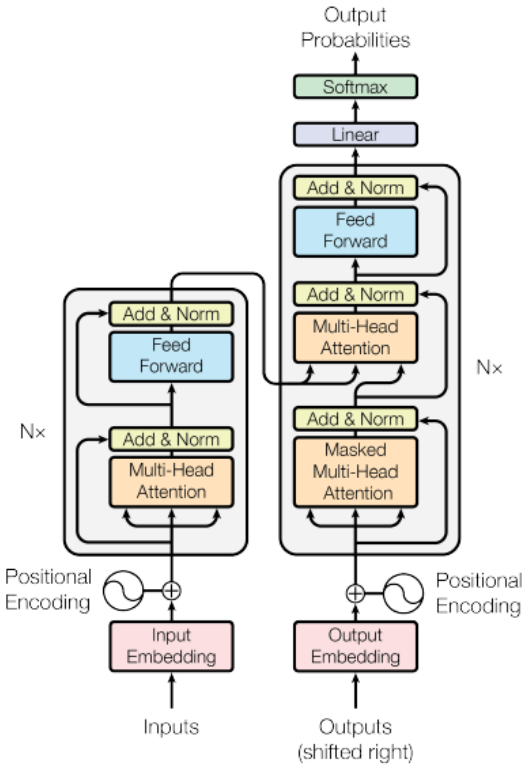

Figure 17: The transformer block architecture. Embedded input in connected to positional encoder and sent to the encoder. The encoder is built of Multi head attention and feed forward layers followed by normalization layers. Encoder output is then used by the decoder, which uses cross attention mechanism to output probabilities after a linear layer.

Self-attention mechanism

The self-attention mechanism is a crucial component of transformers and plays a fundamental role in capturing relationships between different elements within a sequence. In the context of vision transformers, self-attention allows the model to capture long-range dependencies and spatial relationships between image patches.

At its core, the self-attention mechanism calculates the importance or relevance of each element in a sequence to all other elements. In the case of vision transformers, the input sequence is typically a set of image patches. The self-attention mechanism computes attention weights for each patch, determining how much attention should be given to other patches when encoding the representation of a particular patch.

To understand how self-attention works, let's consider a simplified example. Suppose we have an input sequence of image patches, where each patch is represented as a feature vector. For each patch, the self-attention mechanism learns to assign weights to all other patches based on their similarity. The similarity is measured by computing dot products between feature vectors followed by a SoftMax operation to obtain normalized attention weights.

These attention weights indicate the importance of each patch in relation to other patches. Higher weights imply stronger connections, while lower weights suggest weaker connections. The self-attention mechanism then combines the weighted representations of all patches, producing a context-aware representation for each patch that considers the information from the entire sequence.

The advantage of self-attention in vision transformers is its ability to capture both local and global relationships. Unlike convolutional layers, which have fixed receptive fields, self-attention can consider long-range dependencies between patches. This enables the model to capture fine-grained details and capture context across the entire image, leading to improved performance in tasks such as image classification and object detection.

It′s important to note that the self-attention mechanism is computationally intensive, especially when applied to large input sequences. To address this, various techniques have been proposed, such as using approximate or sparse attention patterns, to reduce the computational complexity while still maintaining the effectiveness of the self-attention mechanism.

Overall, the self-attention mechanism is a powerful tool in vision transformers, allowing them to model complex relationships and dependencies within image patches. By leveraging self-attention, vision transformers can capture both local and global information, enabling them to excel in a wide range of computer vision tasks, including object detection, image segmentation, and image classification.

Multi-head attention

Multi-head attention is an extension of the self-attention mechanism that allows transformers to capture different types of relationships and attend to multiple aspects of the input sequence simultaneously. It enhances the expressive power of the model by enabling it to learn different representations and capture diverse patterns.

In a multi-head attention mechanism, the input sequence is divided into multiple subsets, or ″heads,″ and each head performs its own self-attention calculation. Each head learns a distinct set of attention weights, allowing the model to attend to different parts of the input sequence and capture different types of information.

The key idea behind multi-head attention is to provide the model with the flexibility to attend to different positions and learn diverse representations. By incorporating multiple

heads, the model can capture both local and global dependencies, as well as capture different types of relationships within the input sequence.

To compute multi-head attention, the input sequence is linearly projected into multiple subspaces, with each subspace associated with a separate head. The self-attention mechanism is then applied independently to each subspace, generating attention weights and weighted representations. These weighted representations from each head are then concatenated and linearly transformed to produce the final output.

The benefits of multi-head attention are twofold. First, it enables the model to capture different types of relationships and attend to different parts of the input sequence simultaneously. This allows the model to effectively capture both local and global dependencies, capturing fine-grained details while maintaining a broader context. Second, it provides a mechanism for the model to learn diverse representations, as each head learns its own set of attention weights. This promotes model robustness and enhances its ability to capture various patterns and features.

In the context of vision transformers, multi-head attention plays a crucial role in enabling the model to capture complex visual relationships and dependencies. It allows the model to attend to different image patches, capture both local and global spatial information, and learn diverse representations that are beneficial for visual understanding and analysis.

It's worth noting that the number of heads in multi-head attention is a hyperparameter that can be tuned. Increasing the number of heads allows the model to capture more fine-grained relationships but comes at the cost of increased computational complexity. Balancing the number of heads is important to ensure a good trade-off between model performance and computational efficiency.

In summary, multi-head attention is a key component of transformers that enhances their ability to capture diverse relationships and attend to different aspects of the input sequence. By incorporating multiple heads, transformers can capture both local and global dependencies, learn diverse representations, and effectively model complex patterns. In the context of vision transformers, multi-head attention plays a vital role in enabling the models to understand and interpret visual information for tasks such as image classification and object detection.

Transformer encoder architecture

The Transformer encoder is a crucial component of the Transformer architecture, responsible for processing the input data and capturing contextual information. It consists of several layers of self-attention and feed-forward neural networks. Unlike traditional convolutional neural networks, the Transformer encoder does not rely on any convolutional or pooling operations, making it highly parallelizable and capable of capturing long-range dependencies effectively.

Positional Encoding:
One unique aspect of the Transformer encoder is the use of positional encoding. Since the model does not have any inherent notion of order or position in the input sequence, positional encoding is added to convey the relative positions of tokens. This encoding enables the model to capture the sequential information crucial for tasks like natural language processing and image understanding.

The positional encoding is typically added as a fixed representation to the input embeddings. It consists of a set of sinusoidal functions of different frequencies and phases, allowing the model to learn the relative positions of the tokens based on these embeddings. By incorporating positional encoding, the Transformer encoder can distinguish between tokens based on their position in the input sequence, enriching the model's understanding of sequential information.

Self-Attention:
Within each layer of the Transformer encoder, self-attention plays a vital role in capturing the relationships between different tokens in the input sequence. Self-attention allows each token to attend to all other tokens in the sequence, and the model learns to assign different weights or importance to different tokens based on their relevance for the given task. This mechanism enables the model to capture long-range dependencies and contextual information effectively.

Feed-Forward Neural Network:
In addition to self-attention, each layer of the Transformer encoder incorporates a feed-forward neural network. This network consists of two linear transformations with a non-linear activation function in between. The feed-forward neural network helps the model capture complex, non-linear relationships between tokens and enables the encoder to transform the representations learned from self-attention into more expressive and higher-dimensional representations. Layer normalization is applied after each sub-layer in the encoder, including the self-attention and the feed-forward sub-layers.

Layer normalization：
Layer normalization is a technique that normalizes the activations of each layer independently. It helps address the issue of internal covariate shift by reducing the distribution shift across the features of the layer. By normalizing the inputs to each layer, layer normalization helps stabilize the learning process and improves the gradient flow during training.

The layer normalization operation computes the mean and variance of the input activations along the feature dimension and then normalizes the activations using these statistics. It introduces learnable scale and shift parameters that allow the model to adapt the normalized activations to the specific requirements of the task.
By incorporating layer normalization in the Transformer encoder, the model benefits from improved stability and convergence during training. It helps alleviate the vanishing gradient problem and allows for more efficient learning.

The combination of self-attention and feed-forward neural networks in the Transformer encoder allows the model to process the input sequence iteratively, layer by layer, gradually capturing more intricate relationships and generating enriched representations of the input data. These representations are then passed on to the Transformer decoder for further processing and generation.

Transformer decoder architecture

The Transformer decoder is responsible for generating the output sequence based on the information processed by the Transformer encoder. It receives the encoded representations from the encoder and utilizes self-attention and cross-attention mechanisms to generate contextualized representations and make predictions.

Self-Attention in the Decoder：
Similar to the Transformer encoder, the decoder also employs self-attention mechanisms to capture the relationships between different positions within the output sequence. The self-attention mechanism allows each position in the output sequence to attend to all other positions, enabling the decoder to incorporate relevant context and generate accurate predictions.

However, there is a slight modification in the self-attention mechanism of the decoder compared to the encoder. The self-attention in the decoder is masked to ensure that positions attending to future positions are ignored during the prediction process. This masking prevents the decoder from relying on future information that it should not have

access to at each decoding step, ensuring autoregressive behavior in generating the output sequence.

Cross-Attention with Encoder Output:
In addition to self-attention, the decoder also employs cross-attention with the output of the encoder. This cross-attention mechanism allows the decoder to attend to the encoded representations from the encoder and incorporate relevant information from the input sequence. By attending to different positions in the encoder output, the decoder can align the generated output sequence with the input sequence, enhancing the coherence and quality of the predictions.

Similar to the self-attention mechanism, the cross-attention in the decoder is also masked to avoid attending to future positions. The masking ensures that the decoder attends to only the relevant positions in the encoder output based on the decoding step, preventing information leakage from future positions.

Positional Encoding in the Decoder:
Just like the encoder, the decoder also utilizes positional encoding to convey the relative positions of tokens in the output sequence. The positional encoding helps the decoder understand the sequential information and generate the output tokens in the correct order.

Through the combination of self-attention, cross-attention, and positional encoding, the Transformer decoder can effectively generate the output sequence based on the encoded representations from the encoder. The self-attention mechanisms enable the decoder to capture dependencies within the output sequence, while the cross-attention mechanisms allow the decoder to incorporate relevant information from the input sequence. This iterative process of attending and generating leads to the generation of coherent and contextually rich output sequences.

It's important to note that the Transformer encoder and decoder work collaboratively, with the decoder attending to the encoder output at different positions and utilizing the encoded information to generate accurate predictions. This collaborative process forms the foundation of the Transformer architecture and its ability to capture long-range dependencies and generate high-quality output sequences.

3. End to end object detection

The traditional approaches to object detection often relied on multi-stage pipelines, where models performed region proposal generation followed by classification and bounding box refinement. However, these pipelines introduced complexities, such as manual feature engineering and heuristic post-processing steps, which hindered their efficiency and effectiveness.

One of the major challenges with the traditional multi-stage approach was the need for post-processing steps, such as Non-Maximum Suppression (NMS), to filter and refine the detected bounding boxes. While these steps were necessary to remove duplicate and overlapping detections, they were additional components in the pipeline that required careful tuning and were prone to introducing inaccuracies. Moreover, these post-processing steps were often not end-to-end trainable, meaning that the optimization process did not directly consider their performance.

In addition to the challenges posed by post-processing steps, another critical issue arises when the output of the model is directly connected to the image space, particularly in models like YOLO. These models often divide the input image into a grid and assign bounding boxes to specific cells within the grid. While this approach allows for efficient detection, it can lead to challenges in assigning accurate labels to the predicted bounding boxes.

The resolution of the output grid can pose difficulties when objects span multiple cells or are situated near cell boundaries. In such cases, it becomes challenging for the model to precisely assign the correct label to the predicted bounding boxes. This uncertainty in label assignment can result in the model outputting average predictions or struggling to precisely localize objects that are partially covered by multiple cells.
To mitigate this issue, higher output resolution can be used, but the immanent problem remains for any chosen resolution, while the latency of the model increase.
Another resolution was suggested in [31] by assigning outputs to labels with OTA (Optimal transport assignment) algorithm, which makes the matching between outputs and labels more flexible. This approach is very effective, but still does not fully mitigate resolution problems.

To address these limitations, the concept of end-to-end object detection emerged as a promising approach. End-to-end solutions aim to tackle the object detection task holistically, optimizing the entire detection pipeline in a unified manner. By jointly optimizing the model's ability to generate accurate bounding boxes and classify objects, end-to-end approaches offer several advantages.

Firstly, end-to-end object detection models eliminate the need for manual feature engineering. Instead, they learn hierarchical representations directly from the data, enabling more effective feature extraction and representation. This reduces the reliance on handcrafted features and allows the model to capture more intricate patterns and contextual information present in the images.

Secondly, end-to-end models mitigate the issues associated with post-processing steps. By jointly optimizing the detection and classification components, these models can incorporate the post-processing logic within the network architecture itself. This integration allows for more precise control over the detection outputs and ensures that the entire system is trained and fine-tuned based on the desired evaluation metrics, such as mean Average Precision (mAP).

Overall, the end-to-end approach revolutionizes object detection by providing a more streamlined and trainable solution. In the following sections, we will explore some of the notable end-to-end object detection models that have made significant contributions to the field. These models encompass a range of techniques, from incorporating transformers to novel architectural designs, and showcase the advancements achieved in the pursuit of accurate and efficient object detection.

Before we delve into the details of DETR (Detection Transformer) and other concepts that have reshaped the landscape of end-to-end object detection, some related work is reviewed.

Related work

To achieve end-to-end detection, many approaches are explored in the previous literature.

Concretely, in earlier research, numerous detection frameworks [19, 20, 21, 22, 23] based on recurrent neural networks attempt to produce a set of bounding boxes directly.

Though they allow end-to-end learning in principle, they are only demonstrated effectiveness on some small datasets and not against the modern baselines.

Several object detectors [28, 29] used the bipartite matching loss. However, in these early deep learning models, the relation between different prediction was modeled with convolutional or fully connected layers only and a hand-designed NMS post-processing can improve their performance.

Learnable NMS methods [24, 25] and relation networks [27] explicitly model relations between different predictions with attention. Using direct set losses, they do not require any post-processing steps. However, these methods employ additional hand-crafted context features like proposal box coordinates to model relations between detections efficiently, while a real end-to-end approach look for solutions that reduce the prior knowledge encoded in the model.

POTO [30] (Prediction-aware One-To-One) proposed a bipartite matching with architecture based only on convolutions, relaying on spatial prior. This approach is relatively cheap, but like in OTA [31] it gets into troubles trying to handle small and dense objects due to the dependency on output resolution.

DETR

DETR (Detection Transformer) [10] represents a paradigm shift in object detection, introducing a new approach that eliminates the need for handcrafted components such as anchor boxes and non-maximum suppression (NMS). Instead, it leverages the power of transformers, a powerful sequence modeling architecture originally introduced for natural language processing tasks, to directly predict objects and their locations in a single feed-forward pass. By adopting the transformer architecture, DETR has overcome several limitations of previous approaches and demonstrated remarkable performance improvements.

One of the main advantages of DETR over YOLO-like models lies in its ability to handle variable numbers of objects without resorting to anchor boxes. Traditional detectors required anchor boxes to encode prior knowledge about object sizes and aspect ratios, which often introduced complexities in model training and limited their adaptability to objects with different scales. DETR, on the other hand, employs a transformer-based encoder-decoder structure that is inherently capable of handling varying object counts, enabling it to detect objects efficiently and accurately regardless of their sizes.

Moreover, DETR introduces a novel concept called the "set prediction" formulation. Unlike YOLO-like models that predict objects at the grid level, DETR treats object detection as a set prediction problem. It models the entire image as a set of objects and uses transformer encoders to capture the global context and relationships between objects. This set-based approach allows DETR to generate predictions without the

spatial constraints imposed by grid cells, resulting in more flexible and context-aware object detection.

Additionally, DETR eliminates the need for post-processing techniques like NMS by formulating object detection as an optimization problem. By using bipartite matching and the Hungarian algorithm [26], DETR directly associates predicted bounding boxes with ground truth objects, avoiding the ambiguity and inefficiency introduced by NMS. This direct alignment simplifies the detection process, reduces the risk of duplicate detections, and enables a more accurate and reliable detection output.

In the following sections, we will delve deeper into the components and mechanisms of DETR to understand its inner workings and explore its remarkable performance in various object detection benchmarks. The schematic architecture of DETR is shown in Figure 18.

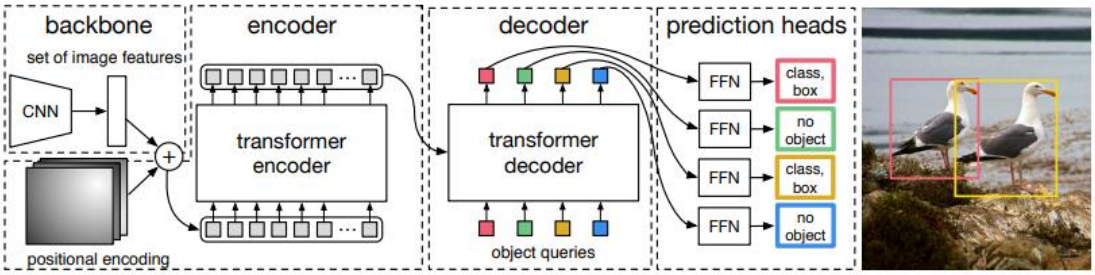

Figure 18: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small, fixed number of learned positional embeddings, which we call object queries, and additionally attends to the encoder output. Each output embedding of the decoder is passed to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.

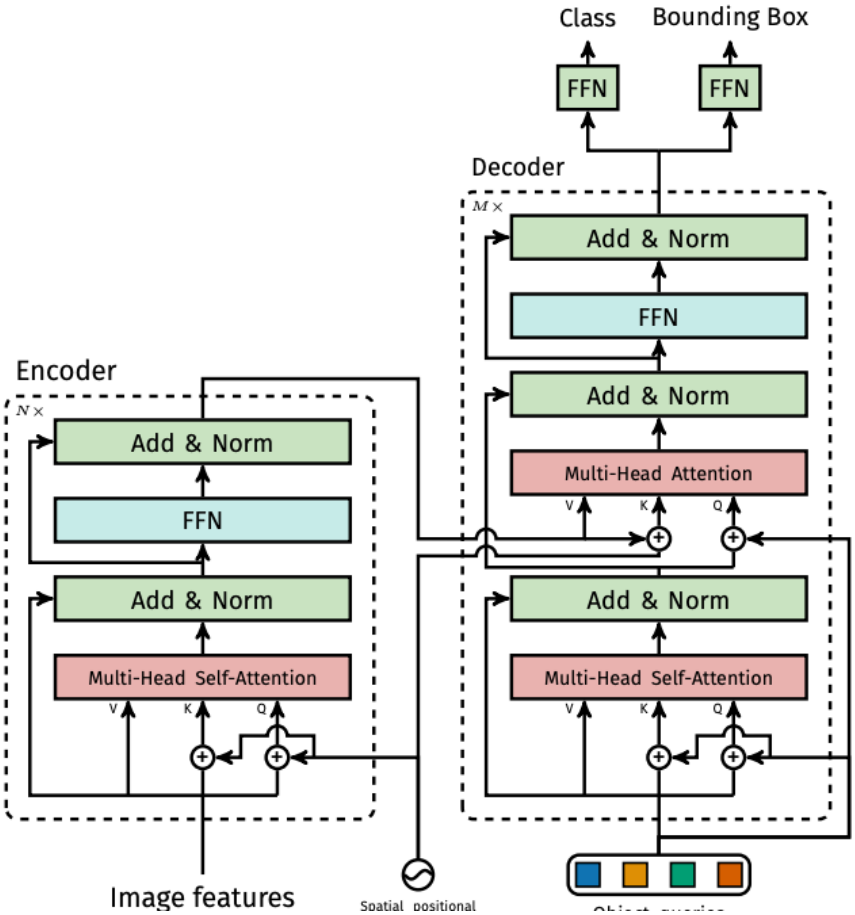

Figure 19: The internal details of Transformers in DETR. Left – the encoder. Right – the decoder and prediction head.

Architectural Components of DETR

DETR introduces a novel architecture that combines the power of transformers with object detection. The model consists of four main components: the CNN backbone, the transformer-based encoder, the transformer-based decoder and finally – predictions head. Together, these components enable DETR to effectively encode and process visual information for accurate object detection.

1. CNN Backbone:

The CNN backbone in DETR plays a crucial role in extracting visual features from the input image. It typically consists of a convolutional neural network architecture, such as ResNet or VGG, that processes the image through a series of convolutional layers to extract high-level visual features. The backbone network serves as a feature extractor, transforming the raw image into a set of feature maps that capture important visual information.

2. Transformer-Based Encoder (Figure 19 left):
The transformer-based encoder is responsible for processing the output feature maps from the CNN backbone. It employs a stack of transformer layers to capture spatial relationships and learn contextual representations. The self-attention mechanism within the encoder allows the model to attend to relevant regions and incorporate global contextual information into the object detection process. By stacking multiple layers, the encoder progressively refines the representations and enables the model to reason about the objects' locations within the image.

3. Transformer-Based Decoder (Figure 19 right):
The transformer-based decoder takes the output of the encoder and performs object detection by predicting the class labels and bounding box coordinates for each object. The decoder utilizes another set of transformer layers to process the encoded features and generate object queries. These queries are then matched with the learned representations of the objects in the image to make predictions.

4. Prediction Head: The prediction head is the final component of the DETR architecture. It takes the refined feature representations from the decoder and produces the final predictions for object detection. The prediction head typically consists of fully connected layers or additional convolutional layers, which map the representations to the desired output format. It outputs the bounding box coordinates and class probabilities for each detected object in the image, when one of the classes is "not object".

In the next section, we will delve deeper into the object detection process in DETR, exploring how it performs set prediction using the transformer-based architecture. We will discuss the unique design choices, loss functions, and training procedures that contribute to DETR's impressive performance in object detection tasks.

Object detection set prediction loss

In DETR, the objective is to generate a fixed-size set of predictions, referred to as N, in a single pass through the decoder. It is important to note that N is deliberately set to be larger than the expected number of objects typically present in an image. During training, one of the key challenges is to accurately score the predicted objects based on their class, position, and size relative to the ground truth.

To address this challenge, DETR utilizes a loss function that facilitates an optimal bipartite matching between the predicted objects and the ground truth objects. This matching process helps establish the best correspondence between the predicted and

ground truth objects. By optimizing this matching, DETR can effectively determine the object-specific losses, particularly those related to the bounding box predictions.

Bipartite matching

Let y be the set of ground truth set of objects (padded with "not object" labels to size N), and $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ the set of the predictions (prediction head's output).
We would like to find a bipartite matching between these two sets, which is formulated as a permutation f the predictions, π, that minimizes some cost function:

$$(1) \qquad \hat{\pi} = \underset{\pi \epsilon \Pi_N}{argmin} \left(\sum_{i=1}^{N} MatchCost\left(y_i, \hat{y}_{\pi(i)}\right) \right)$$

Where $MatchCost$ is a function describes a cost for between the ground truth y_i and the prediction in index $\pi(i)$ (will be defined later).
To obtain this optimal assignment, we can approach it as solving an assignment problem, which can be effectively addressed using the well-known "Hungarian algorithm". In the matching cost calculation, both the class prediction and the similarity between the predicted and ground truth boxes are considered:
We denote $y_i = (c_i, b_i)$, where c_i is the class and b_i is a four elements vector describing the bounding box (width, height and center). Also, for index $\pi(i)$, $\hat{p}_{\pi(i)}(c_i)$ is the probability of class c_i, and $\hat{b}_{\pi(i)}$ the predicted bounding box. Then -

$$(2) \qquad MatchCost\left(y_i, \hat{y}_{\pi(i)}\right) = -\hat{p}_{\pi(i)}(c_i) + L(b_i, \hat{b}_{\pi(i)})$$

Where $L(b_i, \hat{b}_{\pi(i)})$ measures the similarity between the boxes (The lower the better).
Predictions with high probability for the class c_i and similar predicted box, will get a low match cost and will be more likely to match this ground truth box, and vice versa.
Note – the Hungarian matching results a one-to-one assignment, as opposed to YOLO-like models who assign many predictions to one label.
After we computed the best matching $\hat{\pi}$, we define the loss function:

$$(3) \qquad DETRLoss(y, \hat{y}) = \sum_{i=1}^{N} \left[-log\left(\hat{p}_{\hat{\pi}(i)}(c_i)\right) + L(b_i, \hat{b}_{\hat{\pi}(i)}) \right]$$

The calculation of the matching cost follows a similar approach as the definition of the matching cost, with the only difference being the use of the logarithm of the class probability prediction, which has been found to be empirically beneficial.

The last term to be defined is $L(b_i, \hat{b}_{\pi(i)})$, which is the sum of two terms – the L1 loss of every pair of measurements of the boxes, and the 1 – IoU of the boxes. For the padded labels, no bounding box loss is added.

In summary, this novelty loss function encourages the model to output a unique prediction for every label, and a "not object" class for the rest of the predictions.

Performance and Evaluation

To assess the performance of DETR and understand its strengths and weaknesses, it is essential to conduct a comparative analysis with other state-of-the-art object detection models. By comparing DETR with traditional models like RCNN and YOLO, as well as other modern architectures, we can gain valuable insights into its capabilities.

One key aspect to consider is the detection accuracy achieved by DETR compared to other models. DETR has demonstrated remarkable performance in terms of accuracy, often surpassing traditional models. Its ability to handle varying object scales, deformations, and occlusions, combined with the use of transformers for global context modeling, allows DETR to capture intricate object details and achieve high localization precision. Comparative evaluation using standard metrics like mean Average Precision (mAP) shows that DETR (and later – its improved variants) consistently achieves competitive or even superior performance compared to other models. As of the end of 2022 the leaderboard of all the object detection benchmarks is led by variants of DETR.

Another important aspect to evaluate is the computational efficiency of DETR compared to alternative methods. Traditional models like RCNN often rely on complex region proposal mechanisms, which can be computationally expensive. In contrast, DETR's end-to-end nature eliminates the need for region proposal networks and subsequent post-processing steps, resulting in a more efficient inference process. While DETR may have a higher training time compared to some models due to its transformer-based architecture, the inference speed is often faster, making it suitable for real-time applications.

Furthermore, the ability of DETR to handle object detection in a holistic and unified framework brings significant advantages. By simultaneously predicting object classes and their corresponding bounding boxes, DETR eliminates the need for separate classification and localization stages, reducing complexity and potential error propagation. This end-to-end approach enables DETR to deliver more robust and accurate predictions, especially in challenging scenarios with crowded or overlapping

objects. These scenarios, often overlooked by benchmarks that primarily focus on quantitative metrics, highlight the true value of DETR's end-to-end framework. By considering the complete context and interdependencies between objects, DETR can effectively handle complex scenes and produce reliable predictions that capture the intricacies of object relationships.

Limitations and Drawbacks of DETR

While DETR presents a promising approach to end-to-end object detection, it is important to consider its limitations and potential drawbacks. Some notable aspects include:

1. Training Time: Training DETR models can be computationally expensive and time-consuming compared to traditional object detection methods. The large-scale transformer architecture and the bipartite matching process contribute to longer training times, requiring substantial computational resources.

2. Inference Time: Inference with DETR can also be relatively slow compared to other object detection models. The sequential nature of transformer-based processing and the need to process the entire image at once can result in increased inference time.

3. Memory Requirements: DETR models often demand significant memory resources due to the large number of parameters in the transformer architecture. This can pose challenges, particularly when deploying DETR on resource-constrained devices or in real-time applications.

4. Interpretability: The interpretability of DETR can be challenging due to the complex nature of transformer-based architectures. Understanding the decision-making process of DETR and explaining its predictions can be more difficult compared to traditional object detection models.

It is important to consider these limitations and trade-offs when deciding to use DETR in specific applications. While DETR offers significant advantages, addressing these limitations and optimizing the training and inference processes will further enhance its practical utility in various scenarios.

Pix2Seq

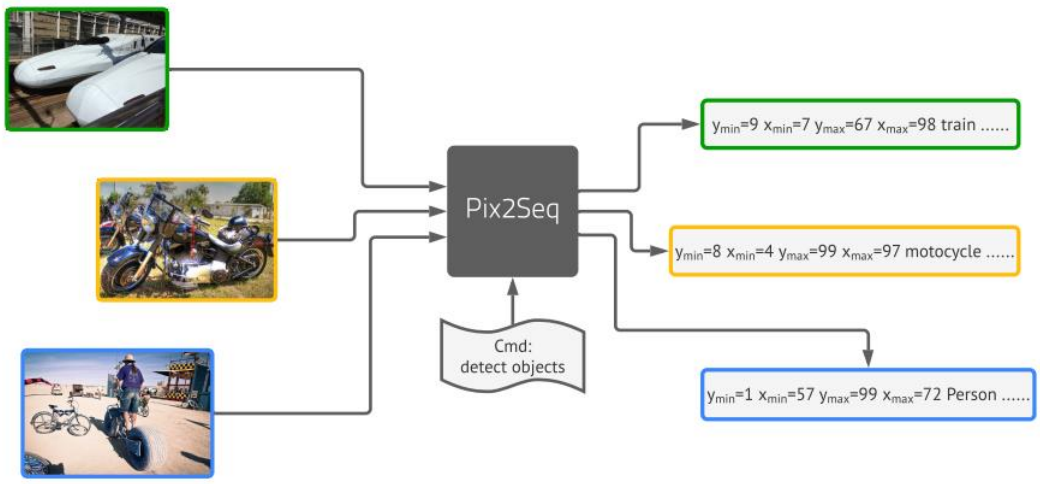

Figure 20: Illustration of Pix2Seq framework for object detection. The neural net perceives an image and generates a sequence of tokens that correspond to bounding boxes and class labels.

"Pix2seq: A Language Modeling Framework for Object Detection" is a new approach to object detection that casts it as a language modeling task (see Figure 20). The authors propose to represent object detection as the task of generating a sequence of tokens that describe the objects in an image. The tokens can be used to represent the bounding boxes and class labels of the objects. The authors then train a neural network to perceive the image and generate the desired sequence of tokens.

Pix2Seq uses a neural network architecture that consists of an encoder and a decoder. The encoder takes the image as input and produces a latent representation of the image. The decoder then takes the latent representation as input and generates the sequence of tokens.

The network is trained using a maximum likelihood objective function. The objective function is simply the probability of the ground truth sequence of tokens given the image.

The authors evaluate their approach on the COCO dataset and show that it achieves competitive results to existing object detection methods. They also show that their approach can be used to perform object detection in real time.

Architecture

Pix2Seq adopts an encoder-decoder architecture for its image-to-sequence translation task. The encoder component is responsible for perceiving the pixel-level information from the input image and encoding it into latent representations. Various encoder architectures can be employed, including Convolutional Neural Networks (like the ones mentioned at the "Common CNN architectures" section), transformer-based models, or a combination of these approaches.

The generation process in Pix2Seq is facilitated by a Transformer decoder. The Transformer decoder generates one token at a time, conditioned on both the preceding tokens and the encoded representation of the input image. This sequential generation approach eliminates the need for complex and customized architectures found in modern object detectors, such as bounding box proposal and regression. Instead, tokens are generated from a single vocabulary using a SoftMax operation, simplifying the overall architecture and enhancing the model's interpretability.

Loss

In a manner similar to language modeling, the training of Pix2Seq involves predicting tokens based on an image and preceding tokens. This prediction is guided by a maximum likelihood loss function, defined as the negative sum of the log probabilities of the predicted tokens given the input image and preceding tokens:

$$-\sum_{j=1}^{L} \mathrm{w_j} \log \mathrm{P}(\tilde{\mathrm{y}}_\mathrm{j} \mid \mathrm{x}, \mathrm{y}_{1:j-1})$$

Here, x represents the given input image, while y and $\tilde{y}$ are input and target sequences associated with x, and L is the target sequence length. y and $\tilde{y}$ are identical in the standard language modeling setup, but they can also be different (e.g., for augmentations). w_j is a pre-assigned weight for j-th token in the sequence.

Inference

During the inference phase, the Pix2Seq model generates sequences by sampling tokens from the model likelihood distribution $\mathrm{P}(y_j \mid \mathrm{x}, \mathrm{y}_{1:j-1})$. This sampling process can be performed in different ways, when the simplest one is to select the token with the highest likelihood.

The generation of the sequence continues until the EOS (End-of-Sequence) token is generated, indicating the completion of the sequence. Once the sequence is generated, it becomes straightforward to extract and de-quantize the object descriptions. This involves retrieving the predicted bounding boxes and class labels.

Performance and Evaluation

As can be seen in the table below (Figure 21), Pix2Seq has demonstrated competitive performance comparable to that of DETR.

Method	Backbone	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN	R50-FPN	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster R-CNN+	R50-FPN	42M	42.0	62.1	45.5	26.6	45.4	53.4
DETR	R50	41M	42.0	62.4	44.2	20.5	45.8	61.1
Pix2seq (Ours)	R50	37M	**43.0**	61.0	45.6	25.1	46.9	59.4
Faster R-CNN	R101-FPN	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster R-CNN+	R101-FPN	60M	44.0	63.9	47.8	27.2	48.1	56.0
DETR	R101	60M	43.5	63.8	46.4	21.9	48.0	61.8
Pix2seq (Ours)	R101	56M	**44.5**	62.8	47.5	26.0	48.2	60.3
Faster R-CNN	R50-DC5	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster R-CNN+	R50-DC5	166M	41.1	61.4	44.3	22.9	45.9	55.0
DETR	R50-DC5	41M	**43.3**	63.1	45.9	22.5	47.3	61.1
Pix2seq (Ours)	R50-DC5	38M	**43.2**	61.0	46.1	26.6	47.0	58.6
DETR	R101-DC5	60M	**44.9**	64.7	47.7	23.7	49.5	62.3
Pix2seq (Ours)	R101-DC5	57M	**45.0**	63.2	48.6	28.2	48.9	60.4

Figure 21: Comparing basic CNNs, DETR and pix2seq. According to pix2seq article writers, it gets the AP with smaller number of params.

One notable advantage of Pix2Seq is its output representation as language tokens, which inherently provides a distribution of box dimensions and classes. This distribution enables applications to obtain more detailed information regarding the model's certainty. For instance, it can be beneficial for an autonomous driving model that receives a sequence of images for estimating the physical movement of objects using a Kalman filter.

On the other hand, as an autoregressive architecture, Pix2Seq is relatively slow.
For example, in a crowded scene, the model will iteratively run O(k*n) iterations, where k is the number of boxes params, and n is the number of objects (as opposed to YOLO and DETR who are not dependent on the number of objects).

4. Experiments

Mobileye Global Inc. (ME) is a company developing autonomous driving technologies and advanced driver-assistance systems (ADAS) including cameras, computer chips and software. I am working at ME since 2020 as an algorithm developer, and my team is specialized in neural networks for 3D object detection based on images inputs.
In this section, I will present and compare the results of YOLO and DETR models I trained for vehicle and pedestrian detection in 2D and 3D (a Pix2Seq model is still not stable). The exact architecture, loss and data are ME confidential, and therefore only limited information will be presented.
It's important to note that the networks are designed for success in the task of autonomous driving on Mobileye's chip. Therefore, both achieving the highest level of performance and running time are of utmost importance, as real-time results need to be obtained.

Additionally, some of the experiments are part of a work-in-progress (WIP) process, so the results are temporary and subject to change.

YOLO-like model

The first model we tried to train was a YOLO like architecture. The network is based on a ResNet-52 backbone and another head in a Unet architecture.
For runtime acceleration, the output resolution is 8 times smaller than the input resolution.

The model outputs both 2D and 3D boxes measurements, and a confidence channel – for every "pixel" of the output. We then apply a post process based on NMS with overlap threshold of 0.1, with some more relaxations to decrease the number of output candidates.

After some hyperparameters tuning we converged a model with a very high-performance exceeding AP of 97.1% for reasonably visible closer than 40 meters vehicles (Recall vs Precision can be shown in figure 22).

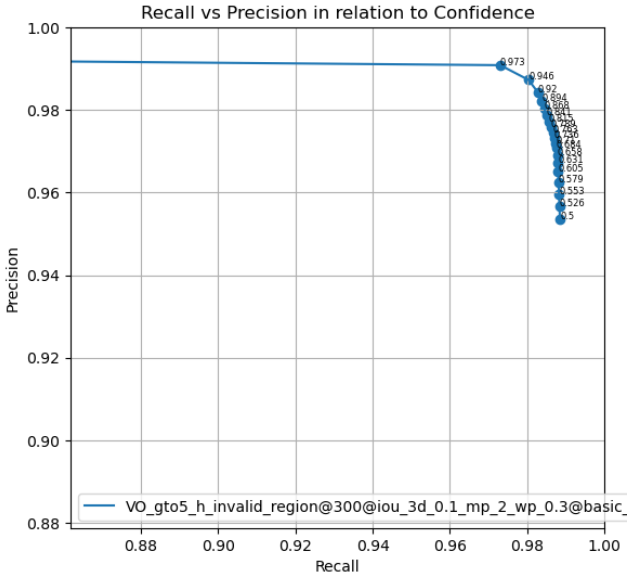

Figure 22: Zoon in of Recall-Precision curve of YOLO model. Results are very good, but still need to be improved.

We chose confidence threshold of 0.946 to achieve recall of 98% for precision of 99%. For harder objects (hidden / far) the AP decreases to 80%-90% and for very hard objects – 60%-70%.

I developed a sophisticated GUI for presenting the raw output of the model and the post processed results (An example is shown in figure 23).

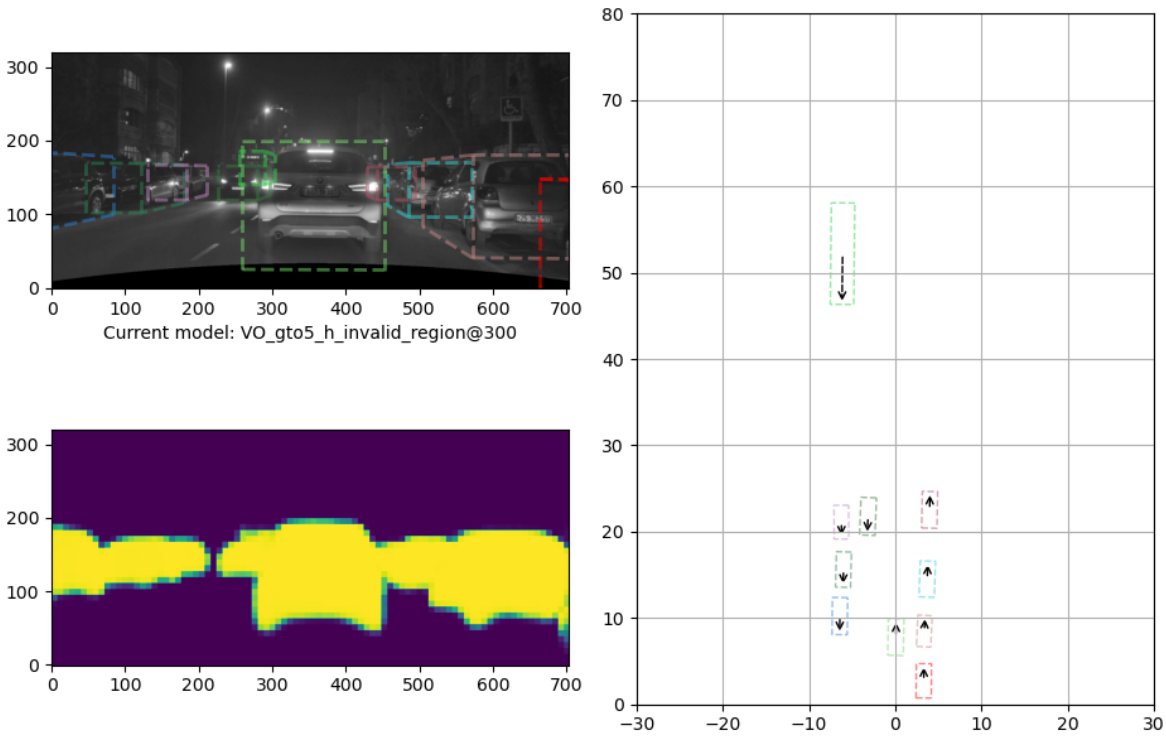

Figure 23: A typical output of YOLO model.

On the top left - input image + model predictions (3d boxes projected to image).

On the bottom left – output confience map.

On the right – birds eye view of 3D boxes prediction

The model and post process are running very fast on EyeQ chip (around 50 ms) and are already in production. Deploying the model to the EyeQ chip was seamless since it is supporting the basic layers such as convolutions and ReLU.

As explained in this paper, some inherent issues popped up -

1. For overlapping objects (in image space), we tend to get "mid-boxes" – both in 2D and 3D (figure 24).
2. Some objects were "detected" by the model but were dropped out because of the sophisticated handcrafted post process.
3. Sometimes NMS clusters two different objects as one and sometimes few predictions of the same object survive it.

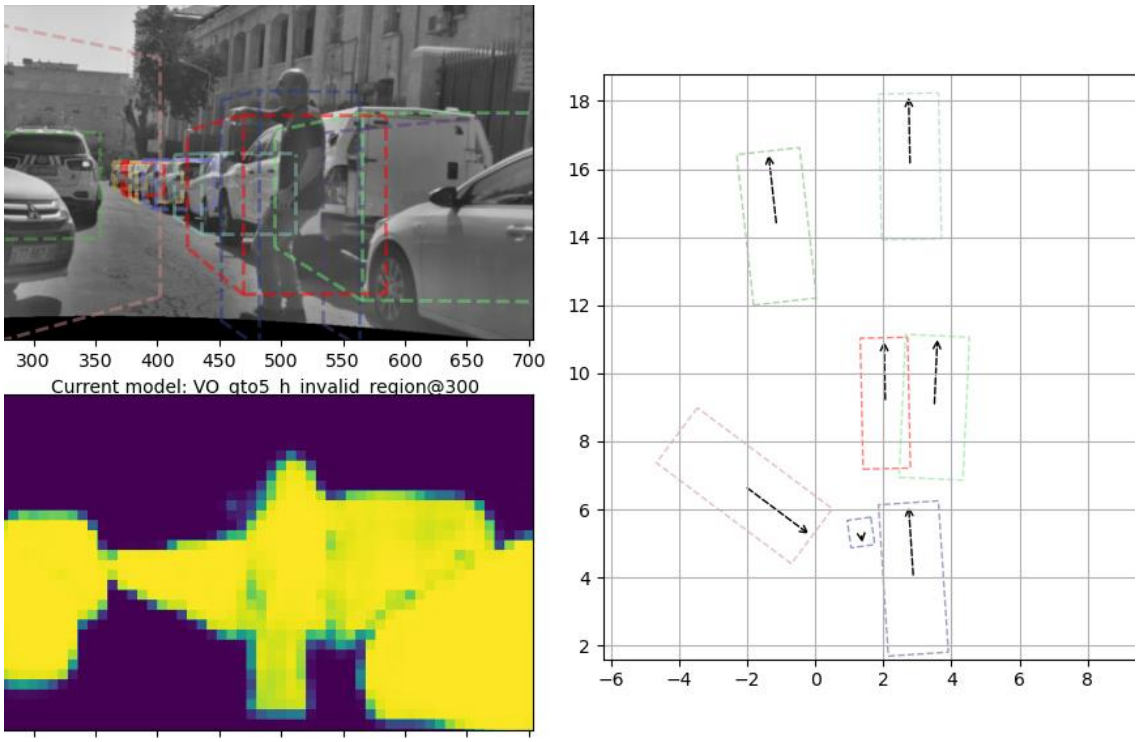

Figure 24: Red box is a false positive caused by the "mid-box issue" – when an output pixel covers two objects in input, prediction tries to capture both objects.

As part of the training process, we have taken measures to avoid cases of mid-boxes and have also adjusted a specific set of post-processing parameters to enhance performance on the validation set. However, these efforts have not completely overcome edge cases that arise across different configurations, resulting in a trade-off between recall and precision in any case.

For example – we trained the model to prefer the closer object when it has a "doubt" whether the pixel in the output belongs to the closer or farer object. This decreased the number of mid-box events, but also decrease the recall on far (and naturally small in image space) objects.

DETR

Based on the same backbone, we switched the YOLO head by a DETR head, and used the set-to-set loss for training.
The first results were hard to converge, so we used some advanced variants of DETR such as DAB (Dynamic Anchor Boxes) [18], which led to great results. In general, DETR model handled hard scenarios better, had less false positives and
The current AP is 97.8% (a bit higher than the YOLO model), and no post process is required.
But moreover – in addition to the better quantitative results, qualitative results are shown as well.
In the following images the first is from YOLO model and the second is from DETR. In the left is the input image (and the output confidence map for YOLO model) and in the right BEV (Bird's eye view). The solid boxes are labels (ground truth), and the dashed boxes are predictions.

1. Line of cars: The objects are dense, and as they get more far, they got less visible and have less pixels in the image. As a result, the YOLO resolution output is not sufficient to cover all the line and many objects are missed, (see Figure 25). As opposed, the DETR model (Figure 26) manage to detect and output more vehicles (though not all of them).

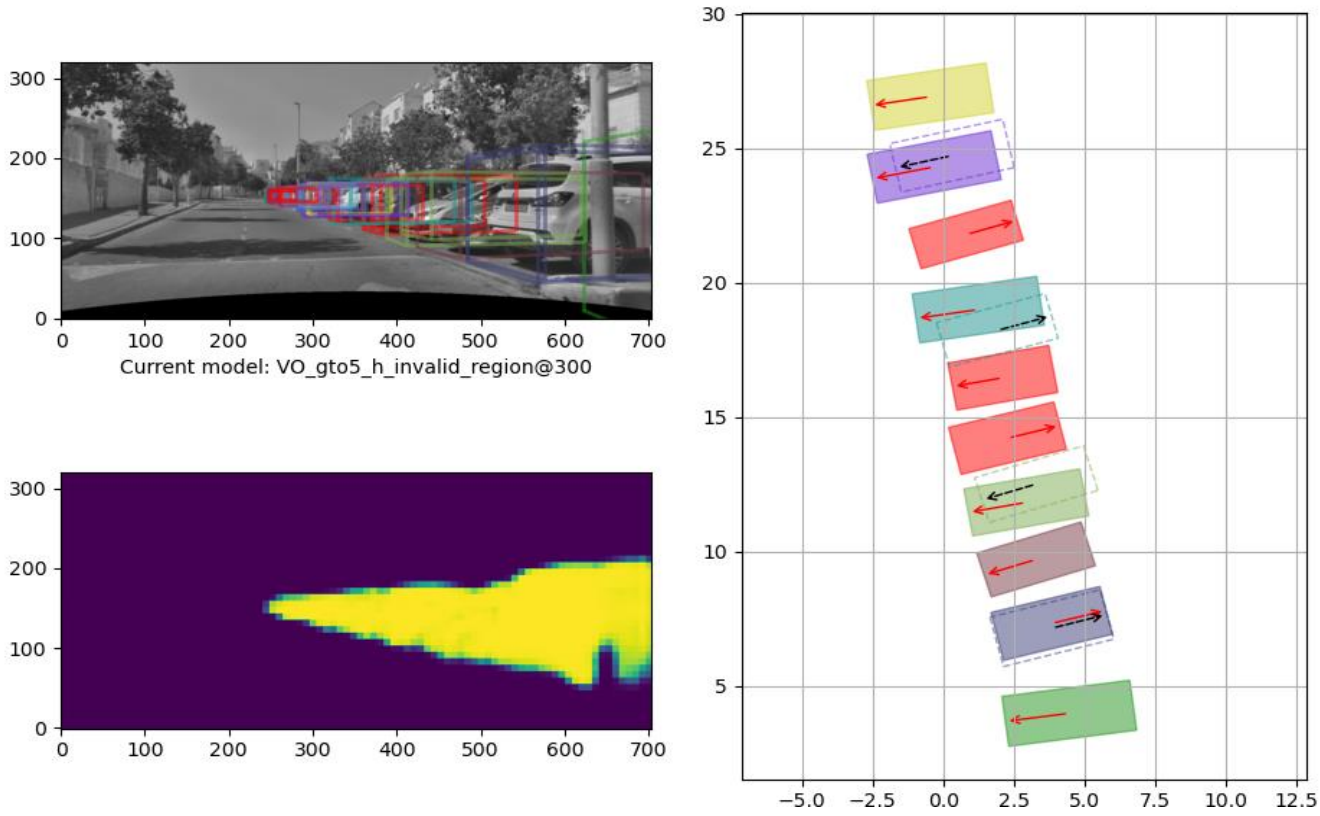

Figure 25: YOLO model misses few objects in the line, missing objects are colored in red.

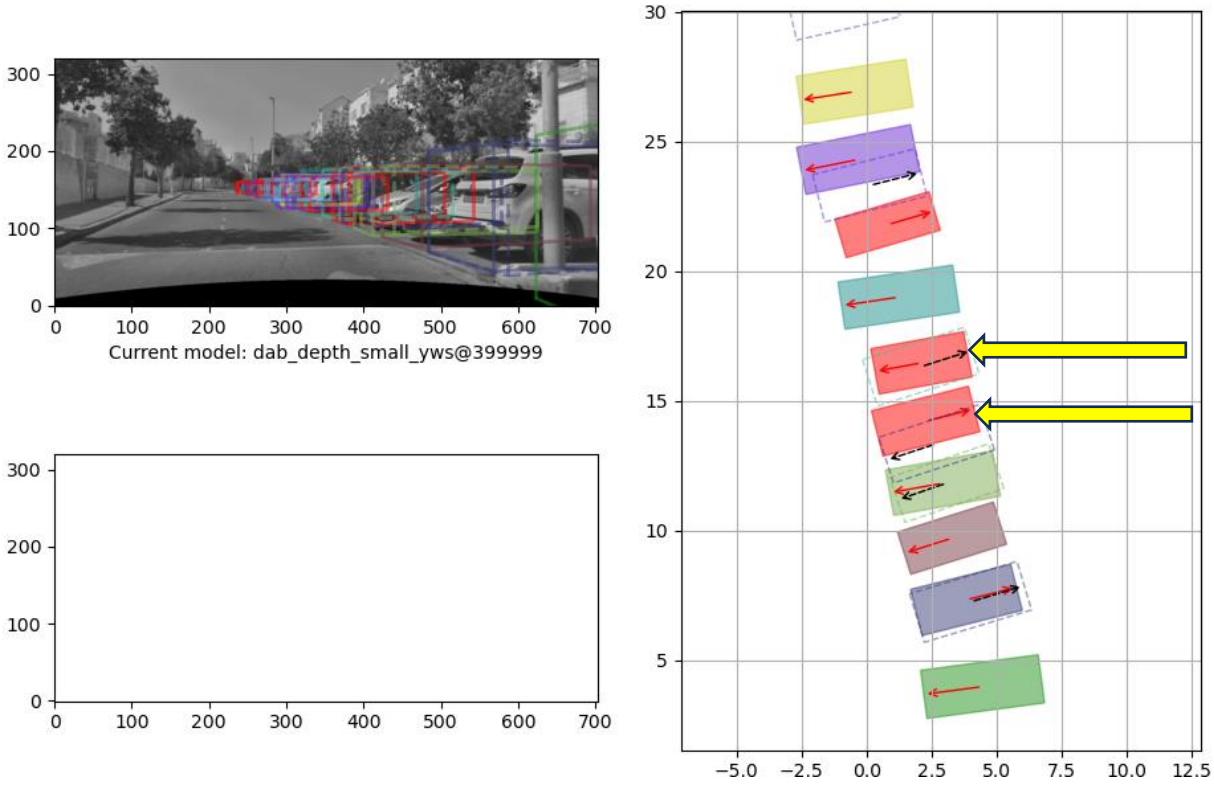

Figure 26: DETR model manages to detect two of the missing objects of YOLO model.

2. Far object: (few pixels in input and output): YOLO model misses the vehicle while DETR detects it:

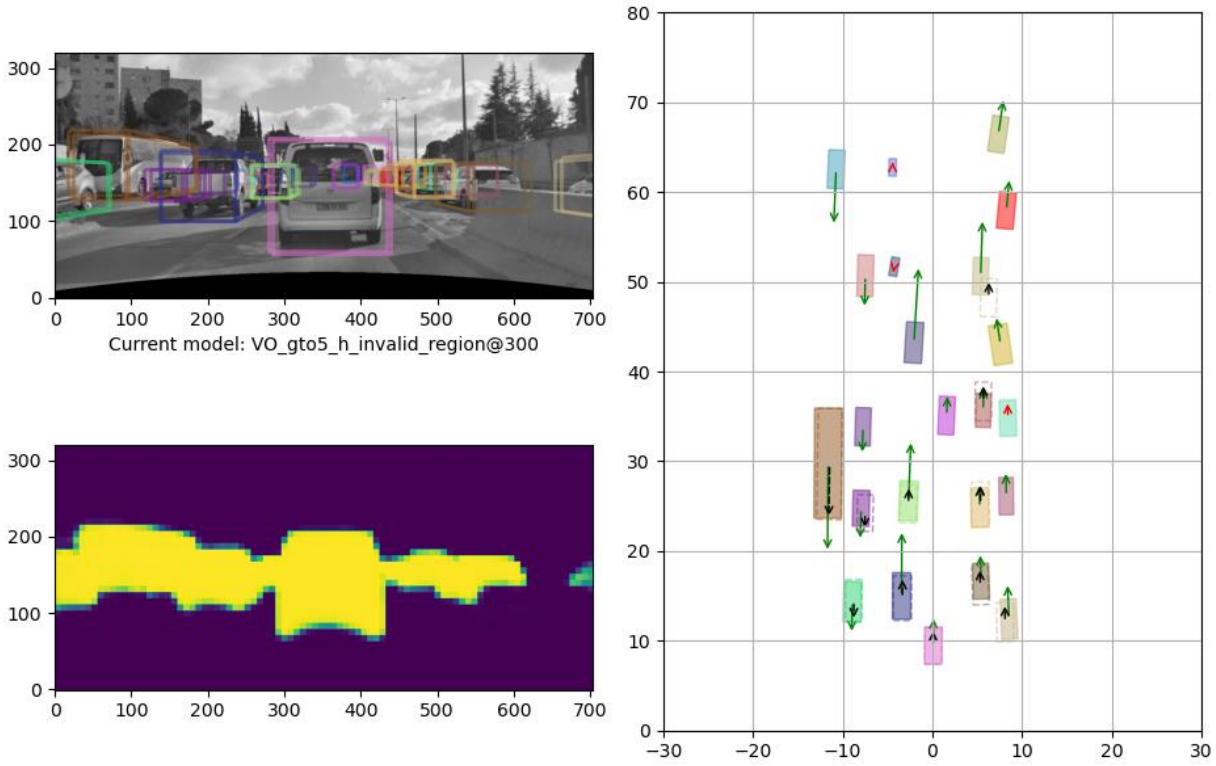

Figure 27: YOLO model misses the vehicle in 60 meters (red colored).

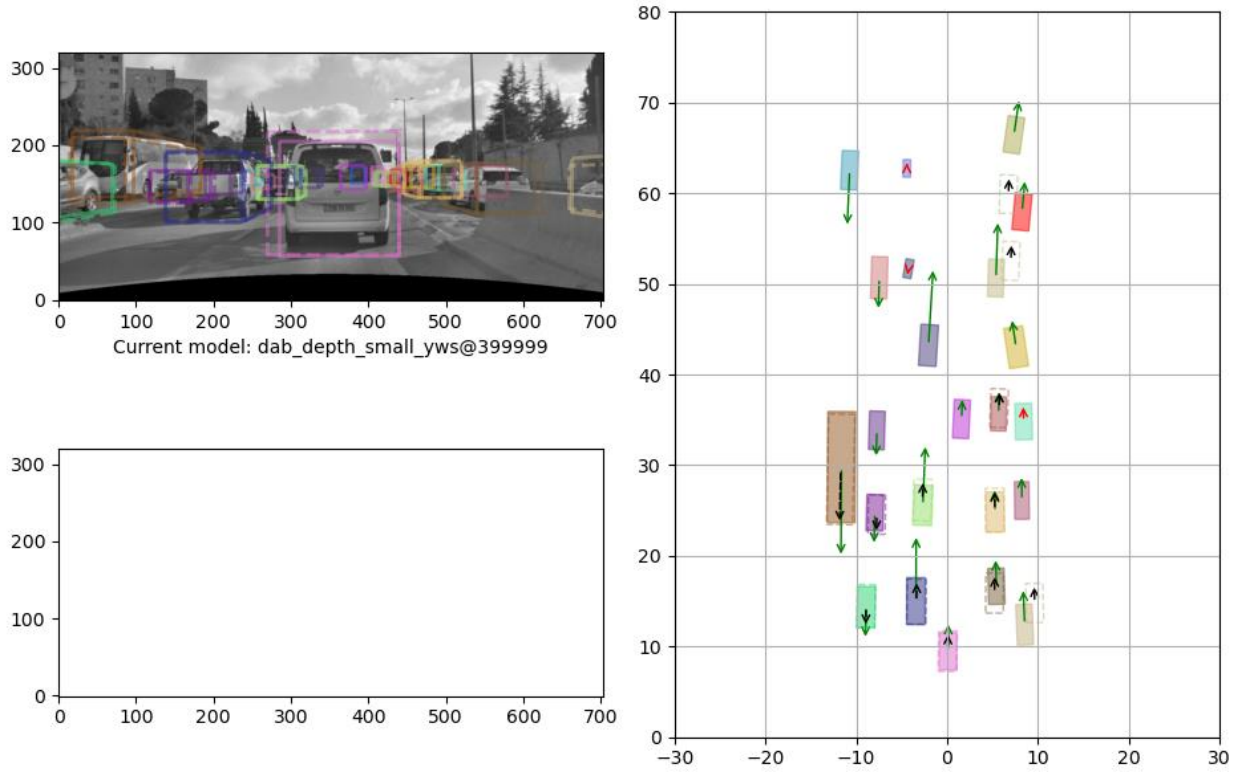

Figure 28: DETR detects the vehicle missed by YOLO, though it's far and hidden.

3. Occluded truck

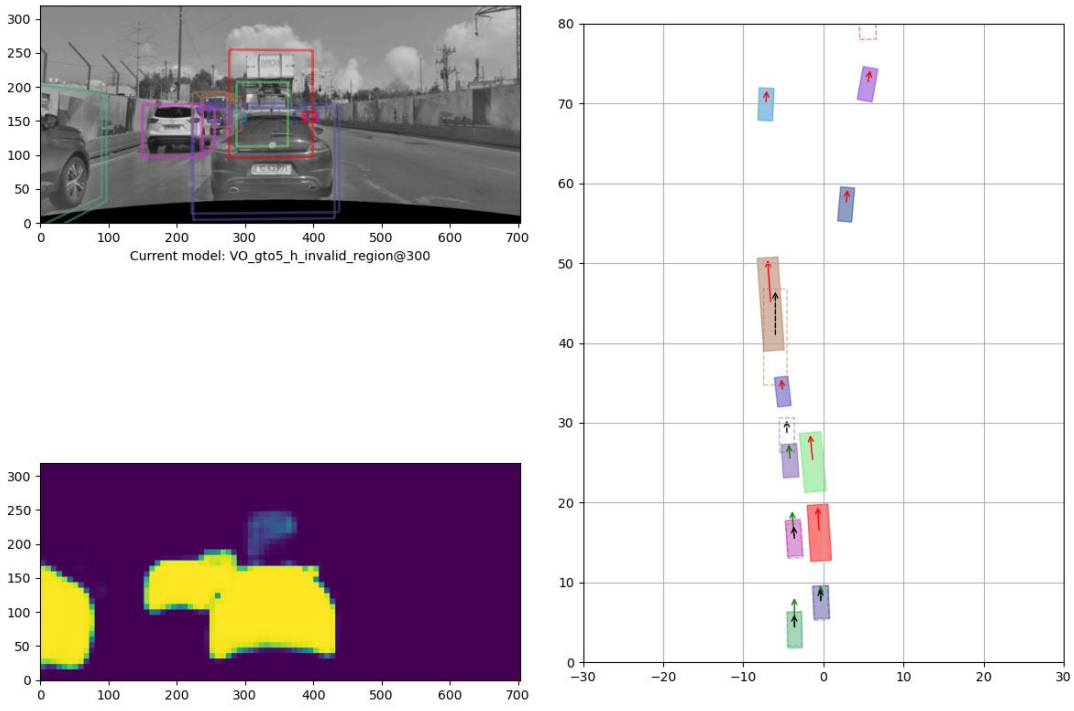

Figure 29: YOLO model misses the truck, hidden by a car.

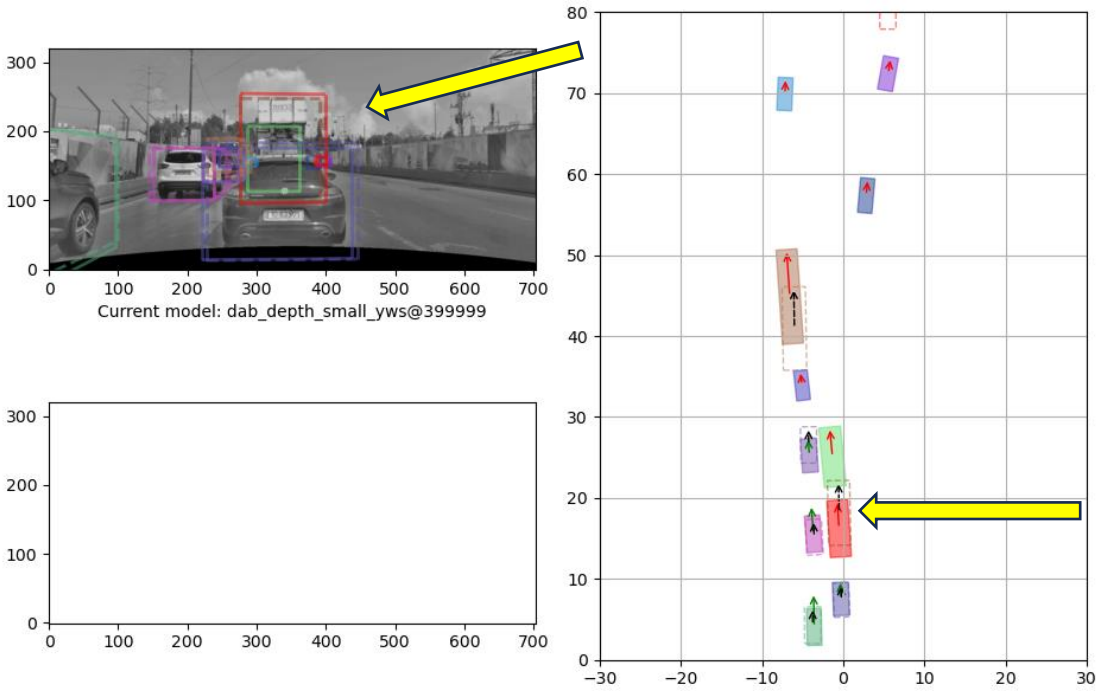

Figure 30: DETR model detects the hidden truck.

The DETR model is a WIP, and we are still trying to improve it. One of the great challenges is to make it train and run (inference) faster. Currently, training DETR model takes around twice the time of YOLO model, and we cannot run it on EyeQ hardware yet since it doesn't support some of the key components of the transformer.

5. Summary and conclusions

In this comprehensive exploration of end-to-end object detection, we have delved into the foundations, methodologies, and cutting-edge advancements in the field. Object detection is a crucial task in computer vision, and over the years, it has seen significant developments, transforming from classical approaches like HOG and Viola-Jones to modern neural network-based models.

We began by elucidating the challenges in object detection, including intra-class variation, the sheer number of categories, and the demand for efficiency in contemporary models. Datasets and evaluation metrics were also discussed, emphasizing the importance of metrics like mean Average Precision (mAP) in assessing detection performance.

We then transitioned to the realm of Convolutional Neural Networks (CNNs) and transformers. We dissected the foundational elements of CNNs, from convolutional layers to common architectures like AlexNet, VGG, and ResNet. The emergence of transformers in the domain of computer vision was explored, showcasing their self-attention mechanism and multi-head attention capabilities.

Moving forward, we introduced state-of-the-art object detection frameworks. The discussion encompassed the RCNN family, followed by the YOLO series. These models have brought remarkable advancements to the field and opened new avenues for real-time object detection.

The notion of end-to-end object detection was highlighted as a significant stride in improving detection accuracy and robustness. By eliminating the need for post-processing steps like Non-Maximum Suppression (NMS), end-to-end models offer a holistic solution, as demonstrated in experiments.

The landscape of object detection has witnessed a remarkable transformation, from traditional methods to contemporary deep learning approaches. With the advent of end-to-end object detection frameworks like DETR, there's a clear trend towards models that integrate object detection and localization seamlessly.

As we conclude this exploration, it's evident that the object detection field is dynamic and evolving. End-to-end models exemplify the potential of modern computer vision techniques. These models not only simplify the detection pipeline but also offer improved accuracy and adaptability.

The future of object detection holds the promise of even more advanced models, novel evaluation metrics, and a deeper integration of machine learning techniques with real-world applications. Whether in autonomous vehicles, surveillance systems, or medical imaging, the quest for precise, efficient, and reliable object detection remains a driving force in computer vision research and development.

6. Bibliography

[1] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Conference on Computer Vision and Pattern Recognition (CVPR).

[2] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 580-587).

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 779-788).

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Hinton, G. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[6] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In European Conference on Computer Vision (ECCV).

[8] PASCAL Visual Object Classes Challenge 2012 (VOC2012). Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. International Journal of Computer Vision (IJCV), 2014.

[9] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[10] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. In European Conference on Computer Vision (ECCV).

[11] Ting Chen, Saurabh Saxena, Lala Li, David J Fleet, and Geoffrey Hinton. Pix2seq: A language modeling framework for object detection. arXiv preprint arXiv:2109.10852, 2021.

[12] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object Detection with Discriminatively Trained Part-Based Models. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 32(9), 1627-1645.

[13] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[14] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems (NIPS), 25, 1097-1105.

[15] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[16] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 1-9).

[17] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 770-778).

[18] Liu, S.; Li, F.; Zhang, H.; Yang, X.; Qi, X.; Su, H.; Zhu, J.; Zhang, L. DAB-DETR: Dynamic anchor boxes are better queries for DETR. arXiv preprint arXiv:2201.12329, 2022.

[19] Eunbyung Park and Alexander C Berg. Learning to decompose for object detection and instance segmentation. arXiv preprint arXiv:1511.06449, 2015.

[20] Mengye Ren and Richard S Zemel. End-to-end instance segmentation with recurrent attention. In IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[21] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. Recurrent instance segmentation. In European Conference on Computer Vision, 2016.

[22] Amaia Salvador, Miriam Bellver, Victor Campos, Manel Baradad, Ferran Marques, Jordi Torres, and Xavier Giro-i Nieto. Recurrent neural networks for semantic instance segmentation. arXiv preprint arXiv:1712.00617, 2017.

[23] Russell Stewart, Mykhaylo Andriluka, and Andrew Y Ng. End-to-end people detection in crowded scenes. In IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[24] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[25] Bodla, N., Singh, B., Chellappa, R., Davis, L.S.: Soft-NMS improving object detection with one line of code. In: ICCV (2017)

[26] Kuhn, H.W.: The hungarian method for the assignment problem (1955)

[27] Hu, H., Gu, J., Zhang, Z., Dai, J., Wei, Y.: Relation networks for object detection. In: CVPR (2018)

[28] Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable object detection using deep neural networks. In: CVPR (2014)

[29] . Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: ECCV (2016)

[30] WANG, Jianfeng, et al. End-to-end object detection with fully convolutional network. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021. p. 15849-15858.

[31] GE, Zheng, et al. Ota: Optimal transport assignment for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021. p. 303-312.