IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 16 April 2024, accepted 26 April 2024, date of publication 15 May 2024, date of current version 24 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3401610

RESEARCH ARTICLE

Quantitative Analysis of Deep Learning-Based Object Detection Models

KHALID ELGAZZAR, (Senior Member, IEEE), SIFATUL MOSTAFI, (Member, IEEE), REED DENNIS, AND YOUSSEF OSMAN

IoT Research Laboratory, Department of Computer, Electrical and Software Engineering, Ontario Tech University, Oshawa, ON L1G 0C5, Canada

Corresponding author: Khalid Elgazzar (khalid.elgazzar@ontariotechu.ca)

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant CRC-2021-00306.

ABSTRACT The rise of convolutional networks in computer vision, especially for generic object detection, has led to the emergence of a myriad of efficient and precise object detection models. Typically, deep learning-driven object detectors operate in two phases: initially, they utilize convolutional networks to extract compact feature embeddings from images; subsequently, these embeddings are used to pinpoint localized object positions. Rooted in convolutional networks, these generic object detection models have the capability to learn from vast datasets that comprise hundreds of thousands of images with thousands of objects. This vast data training gives them unparalleled generalization capabilities, setting them apart from traditional methods. With the swift pace of research, new object detection models are frequently unveiled, each striving for state-of-the-art performance on renowned benchmarks. Given the abundance of viable models, selecting the optimal one can be a daunting task. In this paper, we offer a succinct overview of widely recognized object detectors, emphasizing their architectural distinctions, and presenting a quantitative comparison in terms of accuracy and inference speeds using the popular 2017 Common Objects in Context dataset.

INDEX TERMS Object detection, deep learning, convolutional neural networks, transformers, quantitative analysis.

I. INTRODUCTION

Object detection deals with the localization and classification of existing objects within a given image. Deep learning-based object detection has revolutionized the field of computer vision by significantly advancing the capabilities of automated object identification and localization within images. Traditional object detection methods relied on hand-crafted features and shallow learning algorithms, which limited their ability to handle complex visual data and achieve high levels of accuracy. However, with the advent of deep learning techniques, particularly convolutional neural networks (CNNs), object detection has undergone a transformative shift. Deep learning models can automatically learn hierarchical representations of features directly from raw pixel data, enabling them to capture intricate patterns and relationships within images with unprecedented accuracy.

The impact of deep learning-based object detection extends across various domains, including healthcare, robotics, surveillance, and agriculture. In healthcare, these models facilitate automated analysis of medical images for diagnosis of diseases and treatment planning. In robotics, they enable robots to perceive and interact with their environment, enhancing autonomy and versatility. In surveillance, deep learning-based object detection systems play a crucial role in ensuring public safety and security by detecting and tracking objects of interest in real time. Furthermore, in agriculture, these models aid in crop monitoring, pest detection, and yield optimization, contributing to sustainable and efficient farming practices.

Nowadays, there are many object detection models to choose from and use in applications or research projects. Whilst many models show promising results, they can also be sub-optimal models for different applications. For example, a model can provide extremely high classification and localization accuracy, but due to the nature of the architecture, it also requires a relatively high amount of processing power.

The associate editor coordinating the review of this manuscript and approving it for publication was Chih-Yu Hsu.

Alternatively, some applications may be indifferent to the computational burdens of a model and require only the highest level of accuracy. Some applications might require a trade-off between high accuracy and robust inference speeds.

In this paper, we provide a comparative study between established object detection models, highlighting their architectural and methodical differences. In addition to discussing architectural innovations, we perform evaluation methodologies for popular object detection models and compare their performance in terms of precision and inference speed on the Common Objects in Context (COCO) dataset [1] (a collection of high-quality large-scale datasets for use as a benchmark against generic object detection models). We explore metrics such as precision, recall, average precision (AP), average precision (AP) on different scales, intersection over union (IoU) and inference speed which provide quantitative measures of detection accuracy. Additionally, we examine computational efficiency metrics, including inference speed and model size, which are crucial for real-time applications and resource-constrained environments.

Through a systematic review and analysis of these methodologies, we aim to provide researchers, practitioners, and developers with a comprehensive understanding of quantitative analysis techniques for deep learning-based object detection models. By evaluating the model performance metrics, we seek to empower the research community to make informed decisions in model selection, optimization, and deployment for a wide range of computer vision applications. Our goal is to provide researchers and developers with a concise and easy-to-digest overview of state-of-the-art object detection models, with performance analysis in terms of general accuracy, their performance against different-sized objects, and their inference speeds.

The remainder of the paper is organized as follows. In Section II we review existing studies on deep learning-based object detection techniques. Section III discusses the general structure of a deep learning-based model for computer vision and object detection. In Section IV, we present various state-of-the-art deep learning-based object detectors, providing an overview of their architecture and contributions. In Section V, we present a comparative analysis on their performance on the COCO dataset benchmark based on some key measurements. We finally provide closing remarks and in Section VI.

II. RELATED WORKS

To perform object detection, a model must output the location of each desired object, which is possible via various image processing techniques. For example, edge detection can localize the object by utilizing the pixel intensities of the image. There are certain cases [2], [3] where objects have consistent pixel intensities, allowing for a delineation between a foreground object and the background; potentially denoting the edge of an object. After accumulating all edges or contours of the object, the object can be isolated and detected. This technique, although promising under the right constraints [4], is nontrivial and highly dependent on the object being simple and the image having satisfactory lighting conditions.

For a complete object detection model, a feature extractor can be combined with edge detection, where the first is responsible for acquiring a classification based on the features of the object, while the latter can localize the object. A feature extractor involves extracting information from the image, such as scale-invariant-feature-transform (SIFT) [5], [6], [7] features, that best describe various object classes. Afterward, future images are compared against these feature representations, and if similar, the image is said to contain the respective object class. Numerous other techniques exist that can perform object detection, such as corner detection [8], threshold segmentation [9], and the use of other feature descriptors, such as Speeded-up Robust Features (SURF) [10], SIFT [5], [6], [7], and Histograms of Oriented Gradients (HOG) [11].

All of these feature descriptors suffer from different issues. For example, El-Gayar et al. [12] found that while SIFT is invariant to illumination and affine transformations, it suffers from being slow and providing poor performance with scaling changes. In the same study, it was found that while SURF is faster than SIFT with similar performance, it is unstable when exposed to rotational or illumination changes. In contrast, deep learning techniques provide an end-to-end solution where features are extracted, and objects are localized within the same pipeline. Objects that are under various lighting conditions partially occluded, and generally within different context features can be learned and processed in a fraction of a second on today's hardware. ''Neocognitron'' [13] is one of the first variations of what are today known as Convolutional Neural Networks (CNNs), but it falls short without any sort of supervised learning algorithm. It was not until LeCun et al. [14], [15] showed that backpropagation [16] could be used to train CNNs. CNNs fell out of style shortly after the 1990s as much interest had pivoted towards Support Vector Machines (SVMs). It wasn't until the conception of the ImageNet [17] database, an image database with 3.2 million labeled images, that CNNs started to regain traction. CNNs are a type of deep neural network architected in a way that is optimal for the analysis of any kind of data with a grid-like topology [18]. AlexNet [19] sparked a significant amount of attention towards the application of deep learning for computer vision by providing a model capable of accurate classification within the massive ImageNet dataset, which at the time of its creation consisted of 3.2 million labeled images and has since grown to 14.2 million labeled images at the time of writing this paper. AlexNet showed that CNNs are capable of learning and distinguishing between thousands of classes using supervised learning (training the model on the annotated dataset that contains the image and its respective classification ground truth) and back-propagation. It should be noted that feature selection is completely autonomous in CNNs; all that is provided is the annotated dataset during

the learning phase, and the CNN is completely responsible for all feature processing outside of the pre-processing stage and learning. Object detection models have since leveraged CNNs to generate compact feature embeddings of an image, often using a CNN trained on the ImageNet database. And then further training separate, but connected, CNNs on Localization, Classification, and Segmentation tasks.

Since the early 2000s, there have been a plethora of surveys on the object detection landscape. However, it has not been until recently that deep learning techniques began to accumulate a phenomenal amount of interest from the research community. As such, in the last decade, a large number of surveys have been published focusing primarily on object detection based on deep learning [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. Liu et al. [20] and Jiao et al. [22] provide a very in-depth review of deep learning for generic object detection, covering milestone models such as R-CNN [31], Fast R-CNN [32], Faster R-CNN [33], YOLO [34], [35], [36], [37], [38], [39], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], SSD [49], etc.

Their study provides a very in-depth comparison of various object detection models and backbone networks, including the highlights of their contributions, their advantages, disadvantages, and results on popular benchmarks such as Microsoft COCO [1] and the PASCAL Visual Object Classes (VOC) [50]. They also cover the class-imbalance problem in object detection and discuss various applications where object detection is applied. Masita et al. [21] review the results of multiple research indicating that the use of deep learning in object detection far outperforms traditional approaches that focus on manually created and learned characteristics. Microsoft COCO [1] is known for common objects in Context that use per-instance segmentations to label objects, which helps with accurate object localization. The dataset includes images of 91 different classes that are claimed to be easily recognized by a 4-year-old child. The MS COCO benchmark was produced using unique user interfaces for category recognition, instance spotting, and instance segmentation, with a total of 2.5 million labeled instances in 328k photos. The benchmark provides a thorough statistical analysis by comparing the dataset to PASCAL, ImageNet, and SUN. and presents baseline performance analysis for segmentation detection and bounding box results. The PASCAL Visual Object Classes (VOC) Challenge [50] serves as a benchmark in the recognition and detection of visual object categories, giving the communities of machine learning and vision access to a common dataset of images and annotations, as well as common evaluation methods. From 2005 to the present, the challenge and its related dataset have been held up as the industry standard for object detection. The study also describes the dataset and evaluation process and examines some of the state-of-the-art approaches that have been tested for classification and detection, determining whether they are statistically distinct or not.

Sharma and Mir [25] review the relationship between traditional approaches to object detection, HOG [11], SIFT [5], [6], DPM [51], [52], etc., and deep learning-based approaches. Similarly, [20] and [22] review backbone architectures, commonly used datasets for benchmarking, different proposal methods, and object detection applications. Aziz et al. [26] provides a comprehensive survey of recent advances in visual object detection with deep learning-based object detection models [36], [37], [53] while also maintaining the same pattern as previously discussed papers.

Xiao et al. [28] provides an empirical analysis of various two- and one-stage detectors. The authors also go into significant detail about the common loss functions used to train object detection networks.

III. DEEP LEARNING-BASED OBJECT DETECTION

Methods for deep learning-based object detection usually fall into two categories: region-based proposal (i.e., often referred to as two-stage network) and regression/classification (i.e. often referred to as one-stage networks). Efficient; In both terms of accuracy and performance, object detection is perhaps the most crucial necessity for the success of most intelligent vision-based applications. The biggest advantage of deep learning approaches to object detection is their performance and ability to generalize, while also avoiding the complexity of engineering features by hand that are difficult to design and limited in their ability to be represented in a way that a computer can practically understand.

Convolutional Networks:

Convolutional Neural Networks (CNNs) are the basis for almost all state-of-the-art object detection/classification algorithms. Designed to prosper (maybe change) when working with a grid-like topology, most of the recent practical computer vision applications involving object detection, classification, and segmentation without the involvement of CNNs.

Convolution is a mathematical operation on two functions that produces a third function which describes how the shape of one is modified by the other. Convolution is described by Eq. 1

$$s(t) = \int x(a)w(t-a)da \tag{1}$$

where in terminology relating to convolutional networks, the function x is the input and the function w is the kernel; s is the feature map. While t does not always represent a variable within a time domain, in the case that it does, you can think of convolution as a weighted average of the function $x(a)$ where a is some free variable. However, we are more concerned with discretized convolution on multidimensional array inputs; thus (1) is rewritten as:

$$\begin{aligned} S(i,j)z &= \sum_m \sum_n I(m,n)K(i-m,j-n) \equiv S(i,j) \\ &= \sum_m \sum_n I(i-m,j-n)K(m,n) \end{aligned} \tag{2}$$

Convolution uses three important ideas that help improve the performance of CNNs. These ideas are sparse inter-

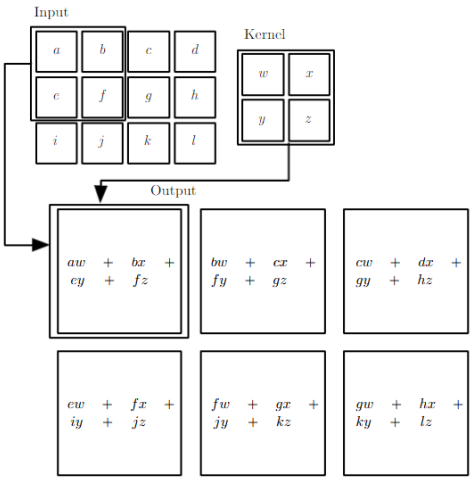

FIGURE 1. Example of basic convolution in CNNs (adapted from [18]).

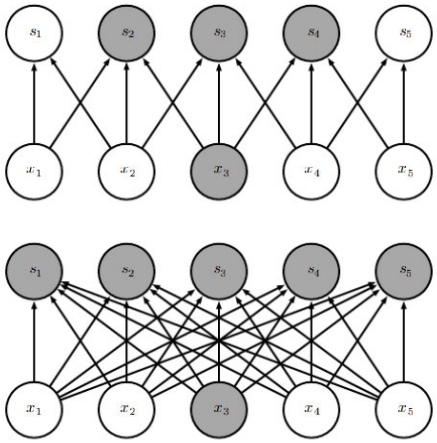

FIGURE 2. The concept of parameter sharing, as you can see when a convolution is performed with a kernel of width 3, only three outputs are affected by one input. This is opposite to the bottom image where convolution isn't performed and all outputs are affected by a single input [18].

actions, parameter sharing, and equivariant representations [18]. This is opposite to regular DNNs, where each output unit interacts with each input unit. The idea of sparse connectivity (interactions) is that instead of using an entire feature set as a densely connected input space, we would convolve over a subset of that feature set with a kernel that is smaller than the input. This allows us to store fewer parameters, thus resulting in better use of memory and it actually increases the statistical efficiency of the model inference [18]. A visual representation of the convolution can be found in Fig. 1.

Parameter sharing involves using the same parameter for more than one node in the network. In a normal deep neural network, each individual weight is used only once when computing the output of an individual layer. The sharing of parameters in convolutional neural networks means that instead of learning a separate set of parameters for every individual location in an image, only one set is needed. This also does not affect the run time, but significantly reduces memory requirements. Two visual examples of parameter sharing can be found in Figures 2-3.

Equivariance is a relatively simple concept. We say that a function $f(x)$ is equivarient to a function $g(x)$ if $f(g(x)) = g(f(x))$. Therefore, the output changes when the input changes.

The final core step to CNN is pooling. A pooling function replaces the output of the network at a certain node with a summary statistic of the nearby outputs [18]. One of the most popular methods of pooling is max pooling. The max clustering reports the maximum output within a rectangular neighborhood [18]. The main advantage of pooling is that it helps make the representation image roughly invariant to small translations in the input image. This means that if we were to nudge the image a small amount to the left or the right, the values of most of the pooled outputs would remain the same. An example of max pooling can be seen in Fig. 4

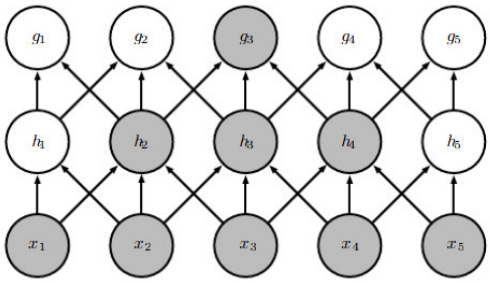

FIGURE 3. An example of the receptive field as a convolutional neural network grows deeper. This displays that even though direct connections in CNNs are sparse, nodes in deeper layers can be indirectly connected to all or most of the input image. Found in [18].

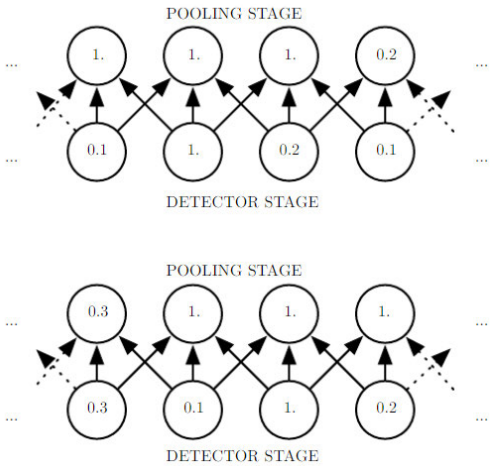

FIGURE 4. An example of how a max-pooling layer would work [18].

In a convolutional network, these steps are generally grouped together into what is called a convolutional layer; with the network itself consisting of thousands of these

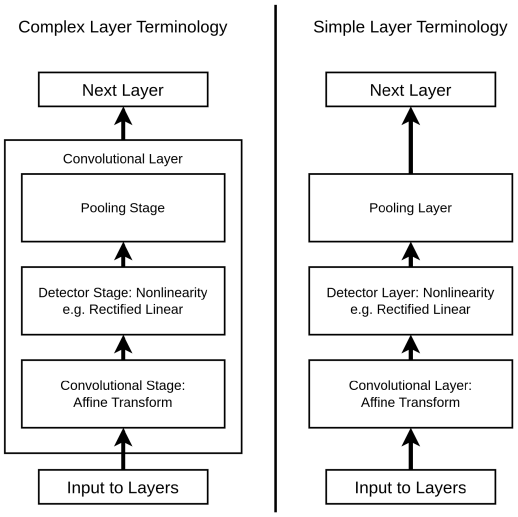

FIGURE 5. An example of a generic convolutional network layer [18].

compositions. An example of a generic convolutional network layer can be seen in Fig. 5.

IV. DEEP LEARNING BASED OBJECT DETECTORS

A. ANCHOR BASED VS ANCHOR FREE DETECTION

This section will discuss anchor-based object detection vs Anchor Free Detection. Some of the popular Ancho based object detection techniques are YOLOv2 [35], YOLOv3 [36], YOLOv4 [37], Scaled-YOLOv4 [38], YOLOv5 [40], PP-YOLO [41], PP-YOLOv2 [42], YOLOR [39], Faster R-CNN [33], FCN [54], RetinaNet [55], SSD [49], EfficientDet [53], etc. Some of the popular Anchor Free Detection techniques are YOLOv1 [34], YOLOX [43], PP-YOLOE [44], YOLOv6 [45], YOLOv7 [46], DAMO-YOLO [47], YOLOv8 [48], CornerNet [33], FCOS [56] etc.

1) ANCHOR BASED DETECTION

Anchor-based object detection was proposed in [33] with the introduction of regional proposal networks. A Region Proposal Network (RPN) takes as input a feature map from a subsequent convolutional layer and outputs several object proposals [33]. In [33] these object proposals were rectangular; each with their own respective objectness score. The objectness score of an object proposal estimates the association between the object proposals class, where there are two categories, object and non-object. This objectness score estimates whether the image segment associated with the object proposal contains a valid object as opposed to the segment belonging to the background class. However, this introduces a class imbalance problem within the detection pipeline as the total number of object proposals containing the background class significantly outnumber those that do not. More information on this is available in [57].

In [33], the authors implemented the RPN as a single convolutional layer and an intermediate fully connected layer that feeds into two fully connected layers. Where the convolutional layer extracted features relevant to the generation of object proposals, the two fully connected layers handled box classification and regression, respectively [33]. A key component of RPNs is reference boxes, or what we often refer to in the literature as anchor boxes (or anchors). In the most basic sense, anchor boxes help the RPN produce more precise object proposals. Every object proposal produced is parameterized relative to several anchor boxes, where the anchor boxes themselves are specifically predetermined in advance at different scales and aspect ratios tailored for generic object detection.

2) ANCHOR FREE DETECTION

Anchor-free object detection does not rely on anchor boxes as reference points for regressing bounding boxes and instead uses alternative methods for learning a model to produce accurate and robust detections. FCOS [56] treated the problem as a per-pixel detection problem, where every individual pixel is responsible for detecting an object. To suppress low-quality bounding boxes, a "centerness" branch was introduced, which was tasked with regressing an estimate for how close to the ground truth center the pixel is. YOLOv1 [35] divided an image into a $S \times S$ grid, each grid is responsible for regressing $B(= 2)$ bounding box predictions, confidences for the boxes, and C class probabilities. CenterNet and CornerNet [58] focus on keypoint detection to regress bounding boxes for objects as shown in. Instead of using anchor boxes as references, CornerNet regresses the corners (pair of keypoints) that encapsulate objects whereas CenterNet regresses the center-points of objects and builds the bounding box around them using basic transformations. In this section, we will discuss established object detectors in research, explaining their architecture and how they function.

B. TWO-STAGE DETECTORS

1) R-CNN

Region-based Convolutional Neural Network (R-CNN), is a pioneering approach in object detection that integrates the strengths of both region proposal algorithms and convolutional neural networks. The primary workflow of R-CNN begins by generating potential bounding box proposals for objects in an image using a region proposal algorithm. Each of these proposals is then resized to a fixed dimension and passed through a pre-trained CNN, typically designed for image classification, to extract feature vectors. Following feature extraction, these vectors are fed into a set of classifiers, usually support vector machines (SVMs), to determine the class of the object within each region. Additionally, a regression model refines the bounding box coordinates to better fit the object. While R-CNN achieved significant improvements in accuracy compared to previous object detection methods, it was computationally intensive due to the need to process multiple region proposals independently. This drawback led to the development of faster variants like

Fast R-CNN and Faster R-CNN, which enhanced both speed and accuracy.

2) FAST R-CNN AND FASTER R-CNN

One of the first ''real-time'' networks was Fast R-CNN [32]. The detection network could achieve inference rates of 0.3 seconds per picture at the time Fast R-CNN was published. Object detectors that use region-based detection, such as R-CNN [31], performed better than any other model to date. The basic concept is to suggest various zones of interest throughout the image and then search for items inside these regions. Fast R-CNN focuses on the latter, creating a Region of Interest (RoI) layer that analyzes possible regions of interest and detects objects inside them.

The model starts the object proposal process with an image and possible regions of interest as inputs. The image's feature map is retrieved initially by sending it via the backbone network until the fully connected and classification layers are reached. Following that, the regions of interest are extracted from the full image's feature map, resulting in multiple feature maps, each pertaining to a distinct region. Every extracted region's feature map is subjected to a max pooling operation, after which the feature maps are shrunk to fixed-length feature vectors. These new feature vectors are smaller and easier to examine while yet preserving significant regional characteristics. Where the first layer classifies the object within the region, the second regresses the bounding box coordinates.

Faster R-CNN improves on Fast R-CNN by incorporating a CNN-based region proposal network (RPN) for region proposal prediction. The Fast R-CNN architecture [32] uses these region suggestions to generate class probabilities and bounding boxes. The RPN creates a collection of RoIs from a feature map generated by a neural layer, each with its own ''objectness'' score. The chance that a RoI includes an object is estimated by its objectness score. The model initially propagates an image through the backbone network, generating a feature map from which the RPN generates region suggestions. A sliding window then moves across said feature map, extracting the feature and transforming them into a fixed length feature vector.

The faster R-CNN then passes the feature vector to two separate fully connected layers. The first layer regresses the bounding box coordinates, while the second classifies the objects included inside the proposed region. An anchoring operation is performed for each window of the original feature map throughout this procedure. Because the RPN's output is region recommendations rather than detections, the output areas of interest are utilised as input for the Fast R-CNN model, which conducts object detection and classification.

3) MASK R-CNN*

Mask R-CNN [59], an extension of Faster R-CNN, was designed, for example, for segmentation tasks. In computer vision, the goal of a segmentation task is to produce a segmentation mask in which each pixel belongs to a specific class.

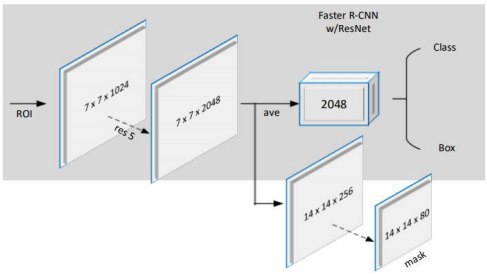

FIGURE 6. Mask-RCNN's model is divided into a detection branch and a segmentation branch.

Consider a picture of a cat standing on grass; each pixel may be classified as cat or grass. Pixels belonging to the cat class would have a value of 0 in the segmentation maps, but pixels belonging to the grass class would have a value of 1. The term ''instance segmentation'' refers to segmenting particular instances within an image rather than the whole image. Mask R-CNN does this by detecting objects and segmenting the image within each detection box.

The architecture of Faster R-CNN, examined in the previous section, is adopted in Mask R-CNN. As highlighted in Fig. 6, the RoI layer extracts the feature map of every proposed region of interest within the image using a ResNet backbone. They send the feature maps to a fully connected layer, which classifies them and regresses their bounding box coordinates. The main distinction in this model is that it conducts segmentation in addition to bounding-box regression through a second, fully convolutional branch. Extracted RoI feature maps are sent into an FCN. As a result, the final output includes the bounding box of the detected item, its categorization, and the mask inside the bounding box. With the inclusion of the mask, Mask R-CNN becomes an all-around framework capable of performing instance-level tasks like microscope image analysis.

C. ONE-STAGE DETECTORS (REGRESSION/CLASSIFICATION)

1) SSD*

For general object recognition, the single shot multi-box detector (SSD) [49] was one of the first to employ a pyramidal feature hierarchy method. The fundamental concept behind this technique is to take a backbone network[1] and add extra convolutional layers that are responsible for both bounding box classification[2] and regression using features obtained from multiple feature maps of varying sizes. As a result, the size of these additional layers gradually decreases, from high- to low-resolution feature maps, with higher-resolution feature maps being responsible for identifying smaller items and

[1]VGG-16 [60] is used in the paper, however, any backbone network is feasible, as the approach is agnostic to the backbone architecture used.

[2]Classification based on the training data being used - not the objectness.

lower-resolution feature maps for detecting larger objects. Each cell in the feature map containing anchor boxes k has been chosen to serve as input for the regression/classification layer. To illustrate, a feature map of size 38×38 would have anchor boxes $38 \times 38 \times k$ in total.[3] In [49] 8732 anchor-box based detections are produced per class in total.

2) YOLOV1 *

You Only Look Once (YOLO) is a class of object detectors first introduced by Redmon et al. [34] in 2016. The main design philosophy of YOLO is to divide an input into a grid and perform the detection task within these grid spaces. This allows the model to forfeit the use of a region proposal network, unlike previous object detectors, which leads to notably higher performance speeds. These real-time rates are accomplished by dividing a picture into a two-dimensional grid with a hyperparameter that determines the grid size (typically 7×7). Every grid cell has a predetermined number of bounding boxes and confidence scores, which denotes the likelihood of a grid cell containing an item and the intersection over union (IoU) of the ground-truth box. Every bounding box has five regressions: the bounding box's center coordinates, width and height, and confidence. The class probability map, which allocates a set of class probabilities to each grid cell, is used to determine the bounding box class. The class with the highest class-specific confidence score is often chosen.

3) YOLO9000 *

For YOLOv2 [35] the authors proposed a novel approach to simultaneously train joint classification and detection models. They used the COCO data set to learn the bounding box coordinates and the ImageNet data set to expand the categories for detection. During training, both data sets were combined, so the detection network was backpropagated with detection training images, and the classification part of the architecture was backpropagated with classification training images. This yielded a YOLO model capable of detecting more than 9000 categories, hence YOLOV2 is also named YOLO9000.

4) YOLOV3 *

As with all object detectors, YOLOv3 [36] needs a backbone to obtain a feature map of the input image. Unlike other models, however, YOLOv3 continues to use the DarkNet architecture, as it presents performance consistent with popular models often employed for feature extraction, but with fewer floating-point operations. Multi-scale object detection is another important feature of YOLOv3. They utilized the last three stages of the DarkNet53 backbone for predictions instead of the feature map retrieved after the final convolutional process. Using numerous outputs from different stages improves detections since larger items are easier to identify later in the process, while smaller objects are better identified early. They, like Faster R-CNN, forecast anchor boxes for the extracted feature maps, preserving the boxes with the highest confidence ratings. Multiple bounding boxes can be regressed for a single item, which is a problem. To deal with this, all predicted boxes are sorted by confidence, and the IoU between the box with the greatest confidence score and the rest of the bounding boxes is calculated. The other bounding box is discarded if the IoU exceeds a certain threshold. This process is called non-maximum suppression.

5) YOLOV4 *

YOLOv4 [37] uses the same anchor-based detection as YOLOv3 and focuses on optimizing other parts of the model. YOLOv4 integrates CSPNet with Darknet53 for a new CSPDarknet53 backbone, adding spatial pyramid pooling (SPP), a path aggregation network (PAN), and a modified spatial attention module (SAM). The SPP module [61], based on spatial pyramid matching [62], takes feature maps from a convolutional layer as input and applies a pooling operator at various spatial sizes. These pooled feature maps are concatenated and used as input for the later layers in the model. This operation has been shown to improve the precision of CNN models. The training process is enhanced by the introduction of new data enhancement approaches. When picture modifications are applied, data augmentation helps to expand and diversify the training data set. Simple augmentation techniques, such as flipping and rotating the image, are utilized. Furthermore, mosaic and Self-Adversarial Training (SAT) are introduced in YOLOv4. Mosaic augmentation combines four random images from the dataset into a single new image, allowing the objects to be seen in diverse contexts and therefore improving the model's performance. SAT augmentation is divided into two stages: In the first, the neural network alters the picture itself rather than weights, and in the second, the network learns this new alteration.

6) YOLOR: YOU ONLY LEARN ONE REPRESENTATION *

YOLOR [39] is a unified network that can encode both implicit and explicit knowledge to complete various tasks through a general representation. This architecture achieves better performance by adding kernel space alignment, prediction refinement, and multi-task learning in the learning process of implicit learning. A unified network is designed to encode implicit knowledge and explicit knowledge together, just as the human brain can learn knowledge from normal learning as well as subconscious learning. The unified network can accomplish multiple tasks simultaneously by generating a unified representation. By incorporating implicit knowledge into the neural network, the model achieves better performance. The unified network provides the ability to understand the physical meaning of multiple and different tasks by utilizing implicit knowledge.

[3]Note that the number of channels does not impact the number of anchor boxes.. It solely relies on a predetermined number of anchor boxes and the $W \times H$ of the feature map.

7) SCALED-YOLOV4 *

CSP approach based YOLOv4 object detection neural network scales both upwards and downwards. It can maintain optimal speed and accuracy and can be applied to small- and large-scale networks. A Network scaling approach modifies not only the depth, width, resolution, but also the structure of the neural network [38]. The YOLOv4-large model achieves a speed of 16 FPS on the Tesla V100 GPU with 55.5% AP (73.4% AP50) for the MS COCO dataset which is state-of-the-art. With test time augmentation YOLOv4-large model achieves 56.0% AP (73.3 AP50). The YOLOv4-tiny model can achieve up to 22.0% AP (42.0% AP50) at a speed of 443 FPS on GPU RTX 2080Ti. YOLOv4-tiny can achieve 1774 FPS by using TensorRT, batch size = 4 and FP16-precision. YOLOv4 is redesigned to a new architecture called YOLOv4-CSP. Scaled YOLOv4 is a further modification of YOLOv4-CSP where the upper and lower bounds of linear scaling models are provided. Factors affecting small and large models are revisited and tuned for model scaling in Scaled YOLOv4 to systematically design YOLOv4-large and YOLOv4-tiny models. These modifications allow Scaled YOLOv4 to reach the most optimal combination of speed and accuracy.

8) YOLOV5 *

The backbone of the YOLOv5 [40] is a customized version of CSPDarknet53, starting with a Stem comprising a strided convolutional layer with a large window size aimed at reducing memory usage and computational demands. Following this, a series of convolutional layers extract pertinent features from the input image. Subsequently, the SPPF (spatial pyramid pooling fast) layer, along with subsequent convolution layers, process these features at various scales, while upsample layers enhance the resolution of the feature maps. The SPPF layer enhances computational efficiency by pooling features of different scales into a fixed-size feature map. Each convolutional operation is accompanied by batch normalization (BN) and SiLU activation. The neck of the model incorporates SPPF and a modified CSP-PAN, while the head structure resembles that of YOLOv3. In YOLOv5, numerous augmentations are employed, including Mosaic, copy paste, random affine, MixUp, HSV augmentation, random horizontal flip, alongside other augmentations from the albumentations package. Additionally, improvements have been made to grid sensitivity to ensure stability against runaway gradients.

9) PP-YOLO *

PP-YOLO [41] (abbreviated PaddlePaddle-YOLO) was a paper released by Baidu researchers in July 2020. At the time, PP-YOLO achieved better results than YOLOv4 with an arguably more simplistic architecture design. The authors replaced the common Darknet53 backbone that was used in the two previous (and most popular) YOLO iterations (v3 and v4, respectively) with a modified ResNet50 backbone. This modified ResNet50 backbone; ResNet50-vd-dcn [41], is based on the ResNet-50-D [63] architecture and training scheme with the added modification of having all 3×3 convolution layers in the last stage (C5) of the network replaced with deformable convolution layers [64]. From the backbone, features are then forwarded to a generic feature pyramid network (FPN), and finally predictions are made using the YOLOv3 head. Aside from exemplifying the feasibility of using a ResNet-based backbone with a FPN neck and YOLOv3 head, PP-Yolo also uses a significant amount of what the authors call ''tricks''. These ''tricks'' consist of data augmentation techniques for training and modifications (or ''injections'' as stated in the paper) to layers of the neck and head of the model.

10) YOLOX *

YOLOX [43] builds upon YOLOV3 by incorporating five significant enhancements: adopting an anchor-free architecture, utilizing multi positives, implementing a decoupled head, refining label assignment techniques, and integrating robust augmentations. Achieving state-of-the-art performance in 2021, YOLOX attains a remarkable balance between speed and accuracy, delivering 50.1% AP at 68.9% FPS on Tesla V100. Its key alterations compared to YOLOv3 include transitioning to an anchor-free approach, employing center sampling for multi positives, implementing a decoupled head for improved task alignment, introducing a simplified label assignment method inspired by the Optimal Transport problem, and incorporating potent augmentations like MixUP and Mosaic. These modifications collectively lead to significant improvements in average precision (AP) compared to YOLOv3.

11) PP-YOLOE *

PP-YOLOE [44] opts not to utilize operators such as deformable convolution and matrix NMS to ensure compatibility across various hardware platforms. Additionally, it boasts the capability to seamlessly adapt to various hardware configurations with differing computing capabilities, enhancing its versatility for widespread deployment. The overall architecture of PP-YOLOv2 comprises a ResNet50-vd backbone with deformable convolution, a PAN neck featuring SPP layer and DropBlock, and a lightweight IoU aware head. In PP-YOLOv2, the ReLU activation function is used in the backbone, while the neck utilizes the mish activation function. Similar to YOLOv3, PP-YOLOv2 assigns only one anchor box for each ground truth object. Furthermore, in addition to classification loss, regression loss, and objectness loss, PP-YOLOv2 incorporates IoU loss and IoU aware loss to enhance its performance.

12) YOLOV6 *

The architecture of YOLOv6 [45] comprises an efficient backbone utilizing RepVGG or CSPStackRep blocks, a PAN topology neck, and a decoupled head using a hybrid

channel strategy. Additionally, the paper introduces advanced quantization methods like post-training quantization and channel-wise distillation, leading to faster and more precise detectors. YOLOv6 demonstrates superior performance compared to previous cutting-edge models in terms of accuracy and speed, including YOLOv5, YOLOX, and PP-YOLOE. A novel backbone named EfficientRep, based on RepVGG, is introduced, employing increased parallelism compared to prior YOLO backbones. The neck utilizes PAN enhanced with RepBlocks or CSPStackRep Blocks for larger models, while an efficient decoupled head is developed following the approach of YOLOX. Label assignment is facilitated using the Task Alignment Learning approach from TOOD, with new classification and regression losses implemented, including a classification VariFocal loss and SIoU/GIoU regression loss. A self-distillation strategy is used for regression and classification tasks, along with a detection quantization scheme utilizing RepOptimizer and channel-wise distillation, contributing to the achievement of a faster detector.

13) YOLOV7 *

YOLOv7 [46] introduced a set of architectural adjustments and enhancements known as ''bag-of-freebies,'' which improved accuracy without sacrificing inference speed, albeit leading to longer training times. One notable enhancement is the Extended Efficient Layer Aggregation Network (E-ELAN), designed to facilitate more efficient learning in deep models by effectively managing gradient paths. YOLOv7 also introduced a concatenation-based model scaling approach, ensuring proportional scaling of block depth and width to maintain optimal model structure. Additionally, YOLOv7 incorporated techniques such as Planned Re-parameterized Convolution and refined label assignment strategies to further optimize training. Furthermore, batch normalization in conv-bn-activation was implemented to streamline the inference process.

14) DAMO-YOLO *

Utilizing the MAE-NAS technique, DAMO-YOLO [47] autonomously identifies efficient architectures. Drawing inspiration from GiraffeDet, CSPNet, and ELAN, AMO-YOLO integrates an Efficient-RepGFPN neck tailored for real-time performance. Acknowledging the effectiveness of a large neck paired with a small head, AMO-YOLO adopts the ZeroHead strategy, retaining just one linear layer for classification and regression tasks. To tackle the challenge of misalignment between classification and regression in dynamic label assignment, AMO-YOLO introduces the AlignOTA approach. This method incorporates focal loss into classification costs and utilizes IoU metrics to assign soft labels, ensuring the selection of aligned samples for each target and resolving the issue holistically.

15) YOLOV8 *

YOLOv8 [48] is built upon YOLOv5's backbone introducing modifications in CSPLayer, now called the C2f module. This module improves detection accuracy by combining high-level features with contextual information. Unlike its predecessors, YOLOv8 adopts an anchor-free model with a decoupled head, allowing independent processing of objectness, classification, and regression tasks, thus improving overall accuracy. The objectness score activation function in YOLOv8 utilizes the sigmoid function, representing the likelihood of objects within bounding boxes, while softmax function indicates class probabilities. Additionally, YOLOv8 integrates CIoU and DFL loss functions, enhancing object detection performance, especially with smaller objects.

16) YOLO-NAS *

YOLO-NAS is designed for real-time edge-device applications, focusing on detecting small objects, improving localization accuracy, and optimizing performance-per-compute ratio. It offers an open-source framework for research purposes. Key features include the introduction of Quantization-aware modules (QSP and QCI) to minimize accuracy loss during post-training quantization, automatic architecture design facilitated by AutoNAC, and a hybrid quantization method that selectively quantizes specific parts of the model. It also adopts a pre-training regimen utilizing automatically labeled data, self-distillation, and large datasets. The AutoNAC system assists users in identifying optimal structures considering factors like data, hardware, compilers, and quantization. Additionally, RepVGG blocks are integrated into the model architecture during the NAS process, resulting in the creation of three architectures—YOLO-NASS, YOLO-NASM, and YOLO-NASL—by adjusting the depth and positions of QSP and QCI blocks.

17) EFFICIENTDET *

EfficientDet [53] aims to tackle two challenges in the object detection landscape, namely efficient multiscale feature fusion and model scaling. The proposed solutions to these two challenges introduced bidirectional cross-scale connections and weighted feature fusion (BiFPN) and joint resolution/depth/width scaling for object detectors. BiFPN took the ideas proposed in [65] and [66] and improved upon them. Concisely, these improvements included the removal of feature layers with minimal contribution to the fusion of features within the network, skip connections from input features (P_4 to P_6) to the bottom-up feature aggregation path (skipping the intermediate top-down layer), and treating each top-down bottom-up path as a single layer within a larger feature network.

The model also learns a weighting mechanism which lets the network determine the contribution an input (feature, channel, or pixel) has on the final output feature map(s). EfficientDet introduced this after realizing that certain features at specific resolutions can contribute more discernible information than others with respect to the output feature map(s).

Compound scaling is applied to the backbone, neck (BiFPN), and head (classification and regression) networks. The depth (number of layers), width (number of channels) and resolution (size of the input image) are scaled according to a predetermined coefficient ϕ. The backbone network is unchanged from [67], that is, EfficientNet-B0 through B6 is used for $\phi \in \{0, 1, \ldots, 6\}$. The depth of the BiFPN network is determined by $3+\phi$, the width of each layer within the BiFPN network was chosen to be $1.35^{\phi} \cdot 64$, where 1.35 was chosen through the grid search as the optimal scaling factor. Finally, the input image resolution is scaled according to $512+\phi \cdot 128$.

18) CORNERNET

CornerNet [58] approached the object detection problem by modeling the encapsulation of an object as a pair of keypoints[4] from which a bounding box would then be constructed. Each pair of keypoints represents the top-left and bottom-right corners of a bounding box containing an object. These key points are encoded using a heat map, $\mathbf{H} \in \mathbb{R}^{H \times W \times C}$, where C is the number of channels (classes). So each index $\mathbf{H}_i \in \mathbb{R}^{H \times W}$, $i = 1, \ldots, C$, encodes a heat map of estimated locations for which a keypoint exists for class i when mapped back to the original input. Indices within the heat map will be highly active if they are estimated to contain a keypoint. Interestingly, heat maps for the top-left and bottom-right corners are learned independently of each other. Since multiple top-left and bottom-right corners can exist for the same class, a method was needed so associations could be made for pairs of keypoints that belong together.

In order to make these associations, the model was learned to regress an embedding for each individual keypoint [58]. If two key points belong together, the respective embeddings for these two individual key points should have a small distance between the two of them. Conversely, key points that don't belong together should have a considerable distance between their respective embeddings.

When attempting to estimate corners for an object, a problem arises in layers of the model with lower resolution (higher stride relative to the input image). The local information diminishes in these lower-resolution layers. Thus, it becomes harder to learn the location of the receptive field relative to an object. Imagine looking at an image through a pinhole without having visualized the entire image. It would be (virtually) impossible to determine exactly where within the image you are looking. One can imagine how this might lead to a problem, especially when your goal is to determine a relative location that will generate a bounding box of best fit; like for a person whose top-left corner would be a slight offset from the 90-degree angle of their head and right/left shoulder. Which, when looking at in 2D space through a pinhole, would inherently look empty - or void of any object. More formally, the lack of local information prevents the model from rationalizing whether a given receptive field is supposed to contain a corner.

As a result, as we move deeper in the model we capture more semantic information at the loss of fine-grained spatial information important for the localization of an object - which is especially important when attempting to predict the corners of objects. In order to overcome this, a pooling layer called ''corner pooling'' [58] is introduced. Corner pooling essentially applies horizontal and vertical max pooling. Where the horizontal max pooling pools from right to left across the entire input space, the vertical max pooling pools from bottom to top. This allows for the model to receive in some measure more global information about the relative location of an object for the current receptive field.

CornerNet also regresses offsets to rectify the loss of precision that can be incurred when mapping the predicted keypoint heat maps back to the original image. This precision loss results from the downsampling of the input space through convolution. As a result, offsets are predicted for each keypoint prior to mapping back to the original image. These offsets are regressed in parallel to the heat maps and embeddings.

19) FCOS

Fully Convolutional One-Stage Object Detection [56] (FCOS) jettisoned the orthodox approach of using anchor boxes for One-Stage object detection frameworks by treating the problem as a per-pixel regression. That is, every pixel (x, y) on a given feature map [65] P_i, $i = 3, \ldots, 7$, is responsible for regressing a bounding box and classifying an object type. Every pixel that falls into a ground truth bounding box is labeled as a positive sample and assigned the respective object class for the given ground truth box; along with the class, a four-dimensional target vector is regressed, where each index of the target vector represents an offset from the given pixel to the sides of the ground truth bounding box. The loss function is the sum of focal loss [55] for class predictions and IoU loss for offset regressions.

The model makes use of an FPN for multi-scale predictions to minimize the number of pixels that intersect multiple ground-truth boxes while also improving the best possible recall of the model as a whole. When it comes to reducing the overlap between ground-truth boxes and individual pixels, each level in the FPN is assigned a hyper-parameter representing the maximum distance that pixels within that feature level are required to regress. If an offset regression for a given pixel satisfies the given threshold $\max(l, t, r, b) > m_i$ or $\max(l, t, r, b) < m_{i-1}$ where, $m_i = [0, 64, 128, 256, 512, \infty]$, then said pixels offset regression is ignored (the pixel is set to a negative sample). E.g. Feature map P_6 would have the bounds [256, 512].

As in [55] the head of the network is shared across all feature levels.[5] One problem with regressing offsets from every single pixel within a ground-truth bound box is that pixels further from the centre of the object produce deplorable

[4] Synonymous with top-left and bottom-right corners.

[5] That is, the same convolutional subnet is applied iteratively to each level within the feature pyramid network.

offset regressions. In order to mitigate this, the authors introduced a single convolutional branch in parallel with the classification branch of the network[6] with the main goal of predicting the centre-ness of a pixel location relative to the ground-truth bounding box. The centre-ness of a pixel $ce_{x,y} \in [0, 1]$ is trained with binary cross entropy loss and is used as a weighting factor (centre-ness × classification score) when ranking the final bounding boxes that are to be chosen (i.e., bounding boxes with a low centre-ness score weeded out through NMS.

20) RETINANET *

RetinaNet [55] (published in 2017) is a fully convolutional one-stage network. RetinaNet has a relatively simple architecture, using a ResNet-X-FPN[7] backbone and a parallel classification and box regression head. The classification and box regression heads are small fully convolutional networks (see Fig. 7) responsible for predicting the probability of the type of object and bounding box location at each spatial position. The regression of the location of the bounding box uses the concept of anchors, which are predetermined (outside of the forward pass of the network) bounding boxes at various locations with varying sizes, the aforementioned spatial positions are merely features at each level of the feature pyramid network.[8] However, while the model itself was not novel and relatively simplistic, the loss function introduced was. This loss function, coined Focal Loss, aimed to remedy the foreground and background class imbalance problem (for a more detailed description refer to [57]). Where the focal loss function $FL(p_t) = -(1-p_t)^{\gamma} \log(p_t)$ assigned more weight to hard examples (negative examples, the classifier fails). Here p_t represents the estimated probability of the one-hot ground truth label. When $\gamma = 0$ the loss function degrades to generic cross-entropy loss; note too that as $p_t \to 0$ the loss function converges to generic cross-entropy loss and when $p_t = 0$ again degrades to generic cross-entropy loss.

21) TRANSFORMERS

Transformer networks, introduced in [69] described a new, groundbreaking architecture for sequence transduction tasks; the transformation of input sequences to output sequences. Sequence transduction tasks were often highlighted within the natural language processing (NLP) domain. Finding use within text-to-speech, speech recognition, and machine translation tasks to name a few. Before the introduction of Transformer networks, all SOTA sequence transduction tasks were presided over by recurrent neural networks (RNNs).

[6]Note: the architecture of the network is identical to that in [55]. Also, when we say in parallel, what is meant is that the two branches share the same input features.

[7]The RetinaNet paper, Focal Loss for Dense Object Detection, doesn't claim a specific depth for the ResNet model used in conjunction with the feature pyramid network. However, common implementations use ResNet-50 and ResNet-101.

[8]Refer to Page 8 of [68] for a more detailed description of what is meant by the spatial position.

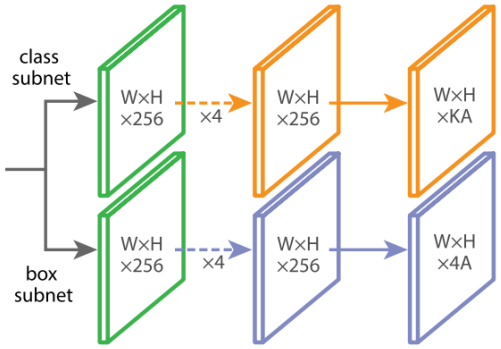

FIGURE 7. Classification and Bounding Box Regression FCN heads used in RetinaNet [55].

The authors of [69] noted that when focusing on these SOTA sequence transduction models, a common pattern emerged; encoder and decoder networks almost always used some kind of attention mechanism [69]. Hence, [69] proposed to eliminate the complexities of recurrences by removing them altogether and rather focus on a network that was based solely on an attention mechanism for generating global relationships between input and output sequences [69].

This Transformer network uses stacked self-attention layers and pointwise fully connected layers for both the encoder and decoder networks [69]. The attention mechanism employed is the *Scaled Dot-Product Attention* (Eq. 3) and *Multi-Head Attention* (Eq. 4).

$$F(Q, K, V) = \sigma\left(\frac{QK^T}{\sqrt{d_k}}\right)V\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3}$$

$$G(Q, K, V) = Concat(h_1, h_2, \ldots, h_n) \cdot W^O \tag{4}$$

$$h_i = F(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \tag{5}$$

where $W_i^{\{Q,K,V\}}$ are learned linear projection matrices with $W_i^Q \in \mathbb{R}^{N \times d_k}$ and $W_i^{\{K,V\}} \in \mathbb{R}^{M \times d_k}$. Note that $Q \in \mathbb{R}^{N \times d_k}$, $K \in \mathbb{R}^{M \times d_k}$ and $V \in \mathbb{R}^{M \times d_v}$, hence $F(Q, K, V) \in \mathbb{R}^{d_m \times d_v}$. The dimensions N, M, d_k, and d_v are hyperparameters, where M represents the number of elements to "attend" and N represents the number of queries. In [69], $N = M = 512$, and $d_k = d_v = 64$.

You can think of Eq. 3 as a weighted sum on all the elements that the network considers, scaled by $\frac{1}{\sqrt{d_k}}$ to counteract the possible saturation of the SoftMax function. And Eq. 4 as intermediate embeddings of the input and output tokens.

Again, we reiterate that while this summary of the Transformer network is believed to provide the reader with enough knowledge to understand the following sections of the paper, it is highly encouraged by the authors that the original paper [69] is read in detail.

a: DETECTION TRANSFORMER

The DEtection TRansformer (DETR) [70] introduced an incredibly straightforward end-to-end object detection framework using transformers [69], capable of on-par performance with the traditional Faster R-CNN framework [33]. Initially,

a CNN backbone would slide across the input image, extracting compact feature embeddings $f \in \mathbb{R}^{C\times H\times W}$. The encoder then applies a 1×1 convolution to f, reducing the C dimension of the feature embeddings to $f_0 \in \mathbb{R}^{d\times H\times W}$. As the transformer architecture expects a sequential input, f_0 is collapsed into a single dimension of $d \times H \times W$. When fed into the attention layer, these feature embeddings are combined with fixed positional encodings, mitigating the permutation-invariance of the transformer architecture, and providing each attention layer with positional information of the input feature embedding. The decoder takes as input the output of the encoder as well as N input embeddings transforming them in parallel at each decoding layer. Akin to the encoder, the decoder is also permutation invariant. As such, for the N input embeddings to produce different results, they must be different. Hence, the N input embeddings are learned positional encodings that are referred to in the literature as *object queries*. These N object queries are added to the output of the encoder and then transformed into output embeddings, which are independently decoded into bounding box coordinates and class labels using a feedforward network, which predicts the normalized centre coordinates, height, and width of the bounding box with respect to the input image, and the class labels using a SoftMax function. This results in N final predictions, where the literature refers to $N = 100$. Note that it is of importance that N is significantly larger than the typical number of objects within an image.

On top of the transformer encoder-decoder, DETR also uses a set prediction loss function via bipartite matching that forces unique matching between predicted and ground truth boxes [70]. The loss function used to train the DETR transformer encoder-decoder is defined by:

$$\mathbb{L}(y,\hat{y}) = \sum_{k=1}^{N}\left[-\log(\hat{p}_{\hat{\sigma}(k)}(c_k)) + \mathbf{1}_{\{c_k\neq\emptyset\}}\mathbb{L}_{\text{bbox}}(b_k,\hat{b}_{\hat{\sigma}}(k)\right]$$

where $y = \left\{y_k = (c_k, b_k) : b_k \in \mathbb{R}^4\right\}_{k=1}^{M}$ denotes the set of M ground truth objects and $\hat{y} = \left\{\hat{y_k}\right\}_{k=1}^{N}$ the set of N predictions made by the transformer encoder-decoder. Note that c_k is the target class label and can take the value of $\emptyset$, i.e., is a background class, or no object. For a given prediction index $\sigma(k)$, $\hat{p}_{\sigma(k)}(c_k)$ represents the probability of class c_k and the predicted bounding box is represented by $\hat{b}_{\sigma(k)}$.

Since DETR approaches the training regime using bipartite matching between y and $\hat{y}$, the lowest cost permutation of N elements $\sigma \in S_N$ needs to be found to train the transformer encoder-decoder. This lowest cost permutation (optimal assignment) is labeled as $\hat{\sigma}$ and defined by

$$\begin{aligned}\hat{\sigma} &= \arg\min_{\sigma\in S_N}\sum_{k=1}^{N}\mathbb{L}_{\text{match}}(y_k,\hat{y}_k)\\ &= \sum_{k=1}^{N} -\mathbf{1}_{\{c_k\neq\emptyset\}}\hat{p}_{\sigma(k)}(c_k) + \mathbf{1}_{\{c_k\neq\emptyset\}}\mathbb{L}_{\text{bbox}}(b_k,\hat{b}_{\sigma(k)})\end{aligned}$$

which is a pair-wise matching cost between the ground truth y_k and a prediction with index $\sigma(k)$ [70]. The Hungarian algorithm is used in practice to compute this lowest cost permutation. $\mathbb{L}_{\text{bbox}}$ is a weighted linear combination of the ℓ_1 and $\mathbb{L}_{\text{iou}}$ = G-IoU loss, and is defined by

$$\lambda_{\text{iou}}\mathbb{L}_{\text{iou}}(b_k,\hat{b}_{\sigma(k)}) + \lambda_{L1}||b_k - \hat{b}_{\sigma(k)}||_1$$

where the weights $\lambda_{\text{iou}}, \lambda_{L1} \in \mathbb{R}$ are hyperparameters, providing the ability to fine-tune which of two loss functions is given more preference towards the overall loss.

b: PVT-{V1, V2}

The Pyramid Vision Transformer (PVT) [71], [72] extends Vision Transformer (ViT) [73] for dense prediction tasks. PVT constructs hierarchical feature maps, a structure that is employed by most CNN backbones, using a Transformer architecture. Doing so in a way that removes the requirements for any sort of end-to-end convolution. Similar to the work done in [73] the input image is treated as a sequence of patches, as such, there is no need for any sort of convolution to extract compact feature embeddings and reduce the dimensionality of the input space.

Since the PVT uses a pyramid-like structure, each stage consists of a similar architecture [71]; a patch embedding layer and Transformer encoding layers. Initially, an input image $I \in \mathbb{R}^{H\times W\times 3}$ is divided into $HW/4^2$ patches. Each patch $P_i \in \mathbb{R}^{4\times 4\times 3}$ is then flattened and linearly projected, producing patch embeddings $P_e \in \mathbb{R}^{\frac{HW}{4^2}\times C_1}$. These patch embeddings are passed through a Transformer encoding layer to produce the initial feature map $F_1 \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times C_1}$, this process is repeated 3 more times, producing feature maps F_2, F_3, & F_4. Where $F_i \in \mathbb{R}^{\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C_i}$. With each feature map reducing in size up to a stride of 32 relative to the original input image by the final feature map F_4. Every transformer encoding stage within this pipeline follows a similar architecture to that in [69], [70], and [73], the number of encoder layers at each level, L_i of the feature pyramid grows linearly with the depth, that is, at the first layer L_1 there is a single encoder layer and at L_4 there are 4 encoder layers.

The PVT [71] does include an addition to the generic multi-head attention mechanism found in [69], [70], and [73], specifically, what the authors refer to as ***spatial-reduction attention***. Spatial-reduction attention (SRA) works similarly to multi-head attention (MHA) but "spatially" reduces the key (K) and value (V) inputs to the multi-head attention mechanism. The intuition behind this addition is to reduce the computation/memory overhead [71] of the model, a common drawback of transformer-based architectures. SRA is defined as follows [71]

$$\begin{aligned}SRA(Q,K,V) &= Concat\,(h_1,h_2,\ldots,h_n)\cdot W^O\\ h_i &= Attention\left(Q\cdot W_i^Q, SR\,(K)\cdot W_i^K, SR\,(v)\cdot W_i^V\right)\\ SR(x) &= Norm\left(Reshape\,(x,R_i)\cdot W^S\right)\end{aligned}$$

$$Reshape(x, R_i);\quad x \in \mathbb{R}^{(H_i W_i)\times C_i} \mapsto x \in \mathbb{R}^{\frac{H_i W_i}{R_i^2}\times (R_i^2 C_i)}$$
$$W_i^Q \in \mathbb{R}^{C_i \times d_k},\ W^O \in \mathbb{R}^{h_i \times d_h},\ W_i^K, W_i^V \in \mathbb{R}^{C_i \times d_h},$$
$$\text{and } W^S \in \mathbb{R}^{(R_i^2 \cdot C_i)\times C_i}$$

The dimension of each head, d_h, is equal to C_i/N_i, where N_i is the number of Transformer encoder heads in the i^{th} stage. R_i denotes the reduction ratio of the attention layers at a given stage i. Note that $Attention(\cdot)$ is equivalent to (3) and W^S is a linear projection that reduces the dimension of the input sequence to C_i [71].

PVTv2 [72] improved the base architecture of PVTv1 [71] by adding overlapping patch embeddings, convolutional feed-forward networks, and linear complexity attention layers [72]. Significantly improve PVT in classification, object detection, and segmentation tasks. The main issues that arose from the original PVT architecture were three-fold. First, when treating the input image as a series of independent sequential patches, it is possible that you might lose some pixel-wise correlations. This is not a problem in CNNs because of the ''growing receptive field'', where deeper layers *may* interact with larger portions of the input space, learning any sort of statistical significance that might exist between pixels. To mitigate this lack of local continuity, PVTv2 uses overlapping patch embeddings to tokenize images. Rather than splitting an image into an SxS grid of equal-sized patches, every patch embedding overlaps with neighboring patches, with zero-padding applied to maintain the same input resolution. In PVTv2 this is done using convolution with zero paddings [72].

The second issue with PVTv1 arises because fixed-sized positional encoding is used to provide encoder layers with positional information of the input feature embeddings. To mitigate this issue, the fixed-size positional encodings in the feedforward layer of the encoder are removed entirely and zero-padded positional encodings via depth-wise convolution are added in place.

The third and final issue with PVTv1 had to do with computational complexity when processing high-resolution inputs, limiting the performance of PVTv1 on most vision-based tasks when compared to traditional CNN-based approaches. To mitigate this problem, PVTv2 replaces SRA with Linear Spatial Reduction Attention (L-SRA), using an average pooling operation, to reduce the spatial dimensionality of the input keys (K) and values (V). Where originally the dimension of the input x was reduced to $x \in \mathbb{R}^{\frac{H_i W_i}{R_i^2}\times C_i}$, L-SRA reduces the dimension of the input x to $x \in \mathbb{R}^{P^2 \times C_i}$, where P is the pooling size of the L-SRA (7 by default) [72].

Similar to [53], [67], [71], and [72] scale the model(s) by modifying the hyper-parameters, a detailed table regarding the values for the hyper-parameters in PVTv1 can be found on page 6 in [71] and for PVTv2 on page 4 in [72].

c: SWIN-T

Swin Transformer [74] or Shifted-Window Transformer is a hierarchical Transformer whose representation is computed with shifted windows. Swin Transformer follows a similar approach to PVT-{v1, v2}, but unlike the initial PVT framework (v1); Swin Transformer operates locally and is able to model the high correlations often found within images. That is, Swin Transformer also constructs hierarchical feature maps using a Transformer architecture.

The Swin Transformer architecture works by first spitting an input image $I \in \mathbb{R}^{H\times W\times 3}$ into non-overlapping patches $P_i \in \mathbb{R}^{4\times 4\times 3}$. A linear embedding, similar to the approach in the PVT architecture, then projects this feature map to an arbitrary dimension C. Swin Transformer blocks are then applied to these patch tokens. The number of tokens is reduced by ''patch merging'' layers as the network grows deeper. In the Swin Transformer architecture, the first patch merging layer concatenates the features of each group of 2×2 neighbouring patches and then applies a linear feed-forward layer to the concatenated features. This reduces the number of tokens by a multiple of 2. Again, similar to PVT the i^{th} stage within the Swin Transformer architecture produces and output shape $O_{S_i} \in \mathbb{R}^{\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C_i}$. Swin Transformer blocks are then again applied to the downsampled features afterward for feature transformation [74]. With multiple stages applied in succession, this creates a hierarchical feature map structure akin to traditional CNN-based architectures.

Swin Transformer blocks replace the traditional multi-head self-attention mechanism utilized in [69], [70], and [73] with an attention module that uses ''shifted windows''. The use of global self-attention as is done in [69], [70], and [73] leads to a quadratic complexity w.r.t the number of tokens. This is an issue for high dimensionality input spaces - a common experience when working within vision. As such, the Swin Transformer architecture applies multi-head self-attention to local $M \times M$ windows containing non-overlapping image patches with a total dimensionality of $h \times w$. This allows for the Swin Transformer architecture to compute multi-head self-attention in a linear-time complexity when M is fixed. To capture the high correlation found within visual signals, successive layers within Swin Transformer blocks shift the window partitioning, where a number of the new window partitions overlap with regions contained by preceding window partitions. Every successive layer within a Swin Transformer block adopts a windowing configuration that displaces the windows by $\left(\lfloor\frac{M}{2}\rfloor, \lfloor\frac{M}{2}\rfloor\right)$ from the initial feature map partitioned into $\left(\lceil\frac{h}{M}\rceil, \lceil\frac{w}{M}\rceil\right)$ windows of size $M \times M$. Note that, in a similar fashion to all other approaches described, when computing self-attention, a relative positional encoding is provided to the input of the $Attention(\cdot)$ operation.

The base model of the Swin Transformer architecture (Swin-B) contains the hyper-parameters of $C = 128$, and the number of layers $= \{2, 2, 18, 2\}$ for Stages 1-4 respectively. Swin-{T, S} simply modifies these hyper-parameters. With

SWIN-S modifying $C = 96$ and SWIN-T modifying $C = 96$ and reducing the number of layers in the third stage from 18 to 6.

V. PERFORMANCE EVALUATION OF DEEP LEARNING BASED OBJECT DETECTORS

We evaluate performance of the state-of-the-art deep learning based object detection models and present a quantitative analysis based on a set of evaluation metrics such as average precision (AP) on different threshold levels of Intersection over Union (IoU), different levels of scales and Inference Speed. Comparison of YOLO-based deep learning object detection models used for generic object detection is presented in Table 1. Comparison of popular deep learning models for generic object detection like Fast-R-CNN, Mask-R-CNN, Cascade Mask R-CNN, RetinaNet, EfficientDet-D, FCOS, FCOS-RT and DetectoRS is presented in Table 2. Transformer-based deep learning object detection models used for generic object detection are presented in Table 3.

A. AVERAGE PRECISION (AP)

Average Precision (AP) in object detection is a commonly and widely used metric to evaluate the performance of an object detection model. AP represents the accuracy of the model in localizing and classifying objects within an image. For each class, the model produces a set of detections along with their confidence scores. These detections are sorted based on their confidence scores. For each confidence threshold, precision and recall are calculated. Precision is the ratio of true positive detections to the total number of predicted positives and recall is the ratio of true positive detections to the total number of actual positives as shown in Equation (6) and (7) respectively. AP provides a comprehensive measure of how well an object detection model performs across various confidence thresholds. Higher AP indicates better performance.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{6}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{7}$$

We computes AP at different threshold of the Intersection over Union (IoU) to evaluate the accuracy of object detection and object localization. IoU is calculated by dividing the area of intersection by the area of union as shown in Equation (8).

$$Recall = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{8}$$

It's expressed as a ratio between 0 and 1. IoU helps measure the overlap between the predicted bounding box and the ground truth bounding box. A high IoU indicates a strong overlap, meaning that the predicted bounding box is close to the ground truth, while a low IoU indicates poor overlap, indicating a significant discrepancy between the predicted and ground truth bounding boxes. In object detection tasks, IoU is commonly used as a threshold to determine whether a detected object is considered a true positive or a false positive. For example, if the IoU between a predicted bounding box and the ground truth bounding box exceeds a certain threshold (often 0.5 or higher), the detection is considered a true positive; otherwise, it's considered a false positive. Here, AP_{bbox} represents AP calculated with an IoU threshold of 1.00. Where as AP_{50} and AP_{75} measures the AP with an IoU threshold of 0.50 and 0.75 respectively.

YOLOv8-640 has the highest AP_{bbox} 59.3% among YOLO-based object detection models as shown in Table 1. EfficientDet-D7X achieves the highest AP_{bbox} which is 54.3% among all other deep learning-based object detection models other than YOLO-based models, as shown in Table 2. Among transformers, Cascade Mask R-CNN with Backbone Swin-S and Swin-B achieved the highest AP_{bbox} 51.9% among transformer-based object detection models as shown in Table 3.

B. AVERAGE PRECISION (AP) ACROSS SCALE

We also compare popular deep learning based object detection models based on Average Precision (AP) across scales to evaluate the performance of a model in detecting objects of various sizes within an image. It's particularly relevant when objects in the dataset exhibit a wide range of scales or sizes. In traditional object detection evaluation, AP is computed using a fixed set of detection thresholds and then averaged over all object categories. However, this approach may not adequately capture the performance of a model across different scales, as objects of different sizes may require different detection thresholds for accurate detection. To address this, we use AP across different scales, which involves evaluating the performance of the model separately for objects of different sizes or scales. We consider subsets of the dataset corresponding to three specific scales small, medium and large objects to compute AP individually for each subset represented by AP_S, AP_M and AP_L respectively. By computing AP across scale, we get insights into how well a model performs for objects of different sizes and identify any potential biases or limitations in its scale-invariance capabilities. This can be particularly important for applications where objects may vary significantly in size or where accurate detection of objects at different scales is critical, such as in aerial imagery analysis or medical imaging.

C. INFERENCE SPEED

Inference speed in object detection refers to the time it takes for a trained model to process an input image and produce predictions about the presence, location, and class of objects within that image. It is a critical metric for assessing the real-world applicability of object detection models, especially in scenarios where timely detection is important, such as in autonomous driving, surveillance, or real-time video analysis. The inference speed is typically measured in terms of frames per second (FPS) or inference time per image as shown in Equation (9). A higher FPS or lower

TABLE 1. A comparison of popular YOLO models used for generic object detection.

Model	Backbone	Anchor	AP_{bbox}	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Inference Speed
YOLO-416	Darknet-24	No	38.4	66.8	40.2	23.3	42.6	53.5	0.012
YOLOv2-416	Darknet-24	Yes	38.4	66.8	40.2	23.3	42.6	53.5	0.012
YOLOv3-416	Darknet-53	Yes	38.4	66.8	40.2	23.3	42.6	53.5	0.012
YOLOv3-512	Darknet-53	Yes	39.9	68.4	41.8	26.8	43.5	52.9	0.014
YOLOv3-608	Darknet-53	Yes	39.1	67.2	41.2	28.5	42.5	49.2	0.017
YOLOv3-Tiny-416	Darknet-53-Tiny	Yes	9.7	19.8	8.4	0.1	7.1	26.7	0.009
YOLOv3-Tiny-512	Darknet-53-Tiny	Yes	9.9	21.1	8.2	0.2	10.9	23.7	0.011
YOLOv3-Tiny-608	Darknet-53-Tiny	Yes	9.5	20.9	7.3	0.4	13.8	19.8	0.012
YOLOv4-416$^{\triangle}$	CSPDarknet53-SPP-PAN	Yes	47.9	70.9	52.7	28.6	53.6	64.4	0.014
YOLOv4-512$^{\triangle}$	CSPDarknet53-SPP-PAN	Yes	50.6	73.8	56.4	33.8	56.5	63.6	0.015
YOLOv4-608$^{\triangle}$	CSPDarknet53-SPP-PAN	Yes	51.3	74.8	57.4	36.6	57	61.9	0.019
YOLOv4-Tiny-416	CSPDarknet53-Tiny	Yes	22.1	40.6	21.8	10.6	26.8	30.8	0.009
YOLOv4-Tiny-512	CSPDarknet53-Tiny	Yes	20.8	39.8	19.5	11	27.8	25	0.010
YOLOv4-Tiny-608	CSPDarknet53-Tiny	Yes	18.7	36.8	17.2	13.8	26.7	19	0.011
YOLOv5-n-640	ModifiedCSPv5	Yes	28.0	45.7	-	-	-	-	0.006
YOLOv5-s-640	ModifiedCSPv5	Yes	37.4	56.8	-	-	-	-	0.006
YOLOv5-M-640	ModifiedCSPv5	Yes	45.4	64.1	-	-	-	-	0.008
YOLOv5-L-640	ModifiedCSPv5	Yes	49.0	67.3	-	-	-	-	0.010
YOLOv5-X-640	ModifiedCSPv5	Yes	50.7	68.9	-	-	-	-	0.012
PP-YOLO-320	ResNet50-vd-dcn	Yes	39.3	59.3	42.7	16.7	41.4	57.8	0.004
PP-YOLO-416	ResNet50-vd-dcn	Yes	42.5	62.8	46.5	21.2	45.2	58.2	0.005
PP-YOLO-512	ResNet50-vd-dcn	Yes	44.4	64.6	48.8	24.4	47.1	58.2	0.005
PP-YOLO-608	ResNet50-vd-dcn	Yes	45.2	65.2	49.9	26.3	47.8	57.2	0.006
PP-YOLOv2-320	ResNet50-vd-dcn	Yes	43.1	61.7	46.5	19.7	46.3	61.8	0.007
PP-YOLOv2-416	ResNet50-vd-dcn	Yes	46.3	65.1	50.3	23.9	50.2	62.2	0.007
PP-YOLOv2-512	ResNet50-vd-dcn	Yes	48.2	67.1	52.7	27.7	52.1	62.1	0.007
PP-YOLOv2-608	ResNet50-vd-dcn	Yes	49.2	68.0	54.1	29.9	52.8	61.5	0.009
PP-YOLOv2-640	ResNet50-vd-dcn	Yes	49.5	68.2	54.4	30.7	52.9	61.2	0.009
PP-YOLOv2-512	ResNet101-vd-dcn	Yes	49.0	67.8	53.8	28.7	53.0	63.5	0.009
PP-YOLOv2-640	ResNet101-vd-dcn	Yes	50.3	69.0	55.3	31.6	53.9	62.4	0.012
YOLOR-CSP-640	CSPDarknet	Yes	50.8	69.6	55.7	31.7	55.3	64.7	0.009
YOLOR-CSP-X-640	CSPDarknet	Yes	52.7	71.4	57.9	33.7	57.1	66.8	0.012
YOLOX-Tiny-416	DarkNet53	No	32.8	67.3	-	-	-	-	0.001
YOLOX-S-640	ModifiedCSPv5	No	40.5	65.4	-	-	-	-	0.003
YOLOX-M-640	ModifiedCSPv5	No	46.9	68.5	-	-	-	-	0.006
YOLOX-L-640	ModifiedCSPv5	No	49.7	69.6	-	-	-	-	0.011
PP-YOLOE-S-640	CSPRepResNet	No	43.0	60.5	46.6	23.2	46.4	56.9	0.003
PP-YOLOE-M-640	CSPRepResNet	No	49.0	66.5	53.0	28.6	52.9	63.8	0.005
PP-YOLOE-L-640	CSPRepResNet	No	51.4	68.9	55.6	31.4	55.3	66.1	0.007
YOLOv6-N-640	EfficientRep	No	35.9	51.2	-	-	-	-	0.001
YOLOv6-T-640	EfficientRep	No	40.3	56.6	-	-	-	-	0.002
YOLOv6-S-640	EfficientRep	No	43.5	60.4	-	-	-	-	0.003
YOLOv6-M-640	EfficientRep	No	49.5	66.8	-	-	-	-	0.006
YOLOv6-L-ReLU-640	EfficientRep	No	51.7	69.2	-	-	-	-	0.009
YOLOv6-L-640	EfficientRep	No	52.5	70.0	-	-	-	-	0.010
YOLOv7-tiny-SiLU	YOLOv7Backbone	No	38.7	56.7	41.7	18.8	42.4	51.9	0.004
YOLOv7-640	YOLOv7Backbone	No	51.2	69.7	55.9	31.8	55.5	65.0	0.006
YOLOv7-X-640	YOLOv7Backbone	No	52.9	71.2	57.8	33.8	57.1	67.4	0.009
DAMO-YOLO-T-416	MAE-NAS	No	42.0	58.0	45.2	23.0	46.1	58.5	0.003
DAMO-YOLO-S-416	MAE-NAS	No	46.0	61.9	49.5	25.9	50.6	62.5	0.003
DAMO-YOLO-M-416	MAE-NAS	No	49.2	65.5	53.0	29.7	53.1	66.1	0.004
DAMO-YOLO-L-416	MAE-NAS	No	50.8	67.5	55.5	33.2	55.7	66.6	0.008
YOLOv8-640	YOLOv8CSPDarknet	No	59.3	76.8	-	-	-	-	0.007
YOLO-NAS-640	NAS	No	52.2	75.6	-	-	-	-	0.006

- We compare the results with batch size = 1
- Inference Speed for PP-YOLO and PP-YOLOv2 are calculated with tensorRT(with TRT)
- $^{\triangle}$ - Uses YOLOv3 head.
- All **YOLO** models are implemented using the Darknet framework.
- $*$ Does NOT include region proposal generation.
- All test are done on the COCO val2017 dataset using a single Nvidia Tesla V100 SXM2 (16GB VRAM) unless specified otherwise.
- Inference speed refers to the average time it takes to process a single image.

TABLE 2. A comparison of popular models used for generic object detection.

Model	Backbone	Anchor	AP_{bbox}	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Inference Speed
Fast R-CNN	ResNet-50-FPN[65]	Yes	37.7	58.9	40.8	22.3	41.2	49.2	0.029*
Faster R-CNN	ResNet-50-FPN	Yes	40.2	61.0	43.8	24.15	43.5	51.9	0.038
Faster R-CNN	ResNet-50-RPN	Yes	38.3	58.6	41.3	20.7	42.6	53.0	0.104
Faster R-CNN	ResNet-101-FPN	Yes	42.0	62.4	45.8	25.2	45.5	54.9	0.051
Faster R-CNN	ResNet-101-RPN	Yes	41.0	61.4	44.0	22.1	45.4	55.8	0.139
Faster R-CNN	ResNeXt-101-32x8d-FPN	Yes	43.0	63.6	46.8	27.2	46.0	54.8	0.098
Mask R-CNN	ResNet-50-FPN	Yes	40.9	61.5	44.9	24.8	43.8	53.3	0.043
Mask R-CNN	ResNet-50-RPN	Yes	39.7	59.5	42.6	22.7	44.5	54.0	0.110
Mask R-CNN	ResNet-101-FPN	Yes	42.9	63.3	46.8	26.3	46.5	56.1	0.056
Mask R-CNN	ResNet-101-RPN	Yes	42.5	62.1	46.0	23.1	47.0	57.6	0.145
Mask R-CNN	ResNeXt-101-32x8d-FPN	Yes	44.2	64.4	48.6	27.5	47.6	56.7	0.103
Mask R-CNN	ResNet-50-FPN-DeConv-C3-C5	Yes	42.6	63.2	46.5	26.8	45.4	56.5	0.047
Cascade Mask R-CNN	ResNet-50-FPN	Yes	44.3	62.2	48.0	26.5	47.2	57.6	0.053
Cascade Mask R-CNN	ResNeXt-152-32x8d-DeConv	Yes	49.2	67.6	53.6	32.0	53.0	63.2	0.234
RetinaNet	ResNet-50-FPN	Yes	38.6	57.9	41.4	23.3	42.3	50.3	0.041
RetinaNet	ResNet-101-FPN	Yes	40.3	60.2	43.1	24.0	44.3	52.1	0.054
RetinaNet	SpineNet-49S	Yes	39.2	58.7	41.9	20.9	42.4	55.6	0.011
RetinaNet	SpineNet-49	Yes	42.1	62.1	45.2	24.2	45.8	58.1	0.015
EfficientDet-D0	EfficientNet-B0-BiFPN	Yes	34.1	52.5	36.0	13.1	39.9	53.7	0.017
EfficientDet-D1	EfficientNet-B1-BiFPN	Yes	40.1	59.0	42.2	21.1	45.9	57.7	0.022
EfficientDet-D2	EfficientNet-B2-BiFPN	Yes	43.4	62.7	46.3	23.7	48.6	60.6	0.029
EfficientDet-D3	EfficientNet-B3-BiFPN	Yes	47.1	66.1	50.5	30.1	51.8	62.6	0.04
EfficientDet-D4	EfficientNet-B4-BiFPN	Yes	49.1	68.6	52.7	32.5	53.6	63.5	0.06
EfficientDet-D5	EfficientNet-B5-BiFPN	Yes	51.1	70.4	55.2	35.5	55.1	65.0	0.108
EfficientDet-D6	EfficientNet-B6-BiFPN	Yes	52.0	71.3	56.0	36.1	56.7	65.7	0.142
EfficientDet-D7	EfficientNet-B7-BiFPN	Yes	53.1	72.4	57.1	36.8	57.3	66.8	0.198
EfficientDet-D7X	EfficientNet-B7-BiFPN	Yes	54.3	73.7	58.5	40.1	57.9	68.0	0.274
FCOS	ResNet-50-FPN	No	41	59.9	44.1	24.9	45.2	51.7	0.039
FCOS	ResNet-101-FPN	No	43.1	62	46.8	27.9	46.9	55	0.051
FCOS	ResNeXt-101-32x8d-FPN	No	43.8	63.3	47.5	28	47.5	55	0.096
FCOS	ResNeXt-101-64x4d-FPN	No	44.7	64	48.1	28.9	48.3	56.7	0.098
FCOS	ResNeXt-101-32x8d-DeformConvV2-FPN	No	46.5	66.1	50.5	29.6	49.4	61	0.125
FCOS-RT	DLA-34-FPN-SHTW	No	38.9	57.3	41.8	21.1	43.6	52.2	0.017
FCOS-RT	DLA-34-FPN	No	39.4	57.4	42.2	22.5	44	53.3	0.019
FCOS-RT	ResNet-50	No	39.2	57.2	42.1	21.3	43.5	53.2	0.021
DetectoRS	Cascade-ResNet-50-FPN	No	47.4	65.6	51.5	29	51.5	62.4	0.124
DetectoRS	HTC-ResNet-50	No	49.1	67.7	53.4	29.9	53	65.2	0.227
DetectoRS	HTC-ResNet-101	No	50.5	69.3	55.3	31.4	54.5	66.8	0.277

- Note that all models presented in this table are tested using PyTorch, CUDA 10.1, and CUDNN 7.6.5.
- $*$ Does NOT include region proposal generation.
- All tests are done on the COCO val2017 dataset using a single Nvidia Tesla V100 SXM2 (16GB VRAM) unless specified otherwise.
- Inference speed refers to the average time it takes to process a single image. Here, inference speed are rounded to 3 decimal places
- *SHTW* means shared towers, I.e. The same FCOS heads are used for every pyramid levels.

inference time indicates faster processing and is generally desirable, as it allows the model to analyze images more quickly, enabling real-time or near-real-time applications. Achieving a balance between inference speed and detection accuracy is crucial when deploying object detection models in real-world applications, as the optimal trade-off may vary depending on the specific use case and requirements.

$$Inference\ Speed = \frac{1}{FPS} \tag{9}$$

Several factors can influence the inference speed of an object detection model. Model architecture: Different object detection architectures have different computational requirements and inference speeds. Some architectures are optimized for faster inference at the expense of slightly lower accuracy, while others prioritize accuracy over speed. The choice of hardware can significantly impact inference speed. GPUs are commonly used to accelerate deep learning inference due to their parallel processing capabilities, but specialized hardware may offer even greater speed improvements. Techniques such as model quantization, pruning, and architecture optimization can reduce the computational complexity of a model, leading to faster inference times without sacrificing accuracy. Processing multiple images with a larger batch size simultaneously can improve GPU utilization and inference speed, but excessively large batch sizes may lead to memory constraints and slower performance. Lowering the input resolution of the images can speed up inference at the cost of reduced spatial accuracy. On the contrary, higher input resolutions may improve detection accuracy, but require more computation and time.

Among all the YOLO-based deep learning models with a AP_{bbox} equal or higher than 52.0%, YOLO-NAS-640 has the lowest inference speed which is 0.006 as shown in Table 1. YOLOv8-640 stands in the second place after YOLO-NAS-640 with an inference speed of 0.007. Although

TABLE 3. A comparison of popular transformer-based models used for generic object detection.

Model	Backbone	AP_{bbox}	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Inference Speed
Mask R-CNN	Swin-T	46.0	68.1	50.3	31.2	49.2	60.1	0.160 (Tesla P100)
Mask R-CNN	Swin-S	48.5	70.2	53.5	33.4	52.1	63.3	0.199 (Tesla P100)
Mask R-CNN	PVT-Tiny	36.7	59.2	39.3	21.6	39.2	49.0	0.068
Mask R-CNN	PVT-Small	40.4	62.9	43.8	22.9	43.0	55.4	0.087
Mask R-CNN	PVT-Medium	42.0	64.4	45.6	24.4	44.9	57.9	0.112
Mask R-CNN	PVT-Large	42.9	65.0	46.6	24.7	45.4	59.4	0.143
Mask R-CNN	PVTv2-b0-FPN	38.2	60.5	40.7	22.9	40.9	49.6	0.065
Mask R-CNN	PVTv2-b1-FPN	41.8	64.3	45.9	26.4	44.9	54.3	0.074
Mask R-CNN	PVTv2-b2-linear-FPN	44.1	66.3	48.4	28.0	47.4	58.0	0.081
Mask R-CNN	PVTv2-b2-FPN	45.3	67.1	49.6	28.8	48.4	59.5	0.099
Mask R-CNN	PVTv2-b3-FPN	47.0	68.1	51.7	30.2	50.4	62.4	0.126
Mask R-CNN	PVTv2-b4-FPN	47.5	68.7	52.0	30.1	50.9	62.9	0.163
Mask R-CNN	PVTv2-b5-FPN	47.4	68.6	51.9	28.8	51.0	63.1	0.175
Cascade Mask R-CNN	Swin-T	50.4	69.2	54.7	33.8	54.1	65.2	0.263 (Tesla P100)
Cascade Mask R-CNN	Swin-S	51.9	70.7	56.3	35.2	55.7	67.7	0.306 (Tesla P100)
Cascade Mask R-CNN	Swin-B	51.9	70.5	56.4	35.4	55.2	67.4	0.359 (Tesla P100)
Cascade Mask R-CNN	PVTv2-b2-linear-FPN	50.9	69.5	55.2	33.6	54.6	65.4	0.138
Cascade Mask R-CNN	PVTv2-b2-FPN	51.1	69.8	55.3	34.8	54.4	66.1	0.155
RetinaNet	PVT-Tiny	36.7	56.9	38.9	22.6	38.8	50.0	0.059
RetinaNet (640x)	PVT-Small	38.7	59.3	40.8	21.2	41.6	54.4	0.052
RetinaNet (800x)	PVT-Small	40.4	61.3	43.0	25.0	42.9	55.7	0.079
RetinaNet	PVT-Medium	41.9	63.1	44.3	25.0	44.9	57.6	0.1058
RetinaNet	PVTv2-b0-FPN	37.2	57.2	39.5	23.1	40.4	49.7	0.054
RetinaNet	PVTv2-b1-FPN	41.2	61.9	43.9	25.4	44.5	54.3	0.065
RetinaNet	PVTv2-b2-linear-FPN	43.6	64.7	46.8	28.3	47.6	57.4	0.074
RetinaNet	PVTv2-b2-FPN	44.6	65.6	47.6	27.4	48.8	58.6	0.092
RetinaNet	PVTv2-b3-FPN	45.9	66.8	49.3	28.6	49.8	61.4	0.120
RetinaNet	PVTv2-b4-FPN	46.1	66.9	49.2	28.4	50.0	62.2	0.157
RetinaNet	PVTv2-b5-FPN	46.2	67.1	49.5	28.5	50.0	62.5	0.169
DETR	ResNet-50-FPN	42.0	62.4	44.3	20.9	45.7	61.0	0.036
DETR	PVT-Small	34.7	55.7	35.4	12.0	36.4	56.7	0.072
DETR-DC5	ResNet-50-FPN	43.5	63.2	46.0	23.1	47.3	61.2	0.083
DETR	ResNet-101-FPN	43.4	63.8	46.2	21.9	47.9	61.8	0.050
DETR-DC5	ResNet-101-FPN	44.6	64.3	47.4	23.0	49.3	62.3	0.097
Deformable-DETR	ResNet-50-FPN	44.5	63.5	48.7	27.0	47.6	59.5	0.066
Deformable-DETR-SS	ResNet-50-FPN	39.3	59.6	42.2	20.6	43.0	55.5	0.037
Deformable-DETR-SS-DC5	ResNet-50-FPN	41.4	61.8	44.9	24.1	45.3	56.0	0.045

- SS stands for Single Scale (i.e., Deformable-DETR-SS would read Deformable Detection Transformer Single Scale).
- DC5 means removing the stride in the C5 stage of the backbone and replacing it with a dilation of 2 instead.

TABLE 4. Recommended models for various tasks.

Model	Backbone	Category
YOLOv8-640	YOLOv8CSPDarknet	$C^{\dagger}$
YOLOX-Tiny-416,	DarkNet53	$C^{\ddagger}$
YOLOv6-N-640,	EfficientRep	$C^{\ddagger}$
YOLOv8-640	YOLOv8CSPDarknet	$C^{\ddagger\ddagger}$
Cascade Mask R-CNN	Swin-S	$T^{\dagger}$
DETR	ResNet-50-FPN	$T^{\ddagger}$
Deformable-DETR	ResNet-50-FPN	$T^{\ddagger\ddagger}$

- $\{C-T\}^{\dagger}$, highest accuracy for transformer based (T) and convolutional network (C) based models
- $\{C-T\}^{\ddagger}$, lowest inference times for T and C based models
- $\{C-T\}^{\ddagger\ddagger}$, best trade-off for T and C based models.

YOLOX-Tiny-416 and YOLOv6-N-640 have the lowest inference speed among all the YOLO-based object detection models, they have a low AP_{bbox} which are 32.8% and 35.9% respectively. Other than YOLO-based object detectors, EfficientDet-D5 has an inference speed of 0.108 with a AP_{bbox} that is equal or higher than 50.0% as shown in Table 2. In this table, RetinaNet having a SpineNet-49S backbone has the lowes inference speed which is 0.011 with a AP_{bbox} of 39.2%. DETR with a ResNet-50-FPN backbone has the lowest inference speed of 0.036 seconds with a AP_{bbox} of 42.0% among all the popular transformer-based object detection models as shown in Table 3. Cascade Mask R-CNN with a backbone of PVTv2-b2-linear-FPN has the lowest inference speed of 0.138 seconds among transformers with a with a AP_{bbox} equal or greater than 50.0%.

D. TRADE-OFF BETWEEN ACCURACY AND INFERENCE SPEED

Table 4 shows the recommended models for various tasks. For convolutional network-based (C) object detection models, YOLOv8-640 with YOLOv8CSPDarknet backbone shows highest accuracy in terms of AP_{bbox} where as the models YOLOX-Tiny-416 with backbone DarkNet53 and YOLOv6-N-640 with backbone EfficientRep shows the lowest inference speed. We recommend YOLOv8-640 with YOLOv8CSPDarknet backbone as the best trade off between the highest accuracy and lowest inference time for convolutional network (C) based object detection models. For transformer-based (T) object detection models, Cascade Mask R-CNN with Swin-S backbone shows highest accuracy in terms of AP_{bbox} where as the models DETR with backbone ResNet-50-FPN shows the lowest inference speed. We recommend Deformable-DETR model with ResNet-50-FPN backbone as the best trade-off between the highest accuracy and the lowest inference time for transformer-based (T) object detection models.

Balancing the trade-off between accuracy and inference speed within object detection models holds significant importance across various domains and applications. In scenarios demanding real-time responsiveness, like autonomous driving or surveillance, swift detection of objects is paramount for ensuring safety and making timely decisions. Here, prioritizing high inference speed is crucial to rapidly process environmental data. However, compromising accuracy for speed could lead to critical errors such as missed detections or false positives, ultimately compromising the reliability and effectiveness of these systems.

In resource-constrained environments, such as edge devices or IoT devices, optimizing inference speed becomes imperative due to limited computational resources. While sacrificing some accuracy might be acceptable to maintain efficiency within these constraints, ensuring that the model can operate effectively on the device is essential. For instance, in mobile applications like augmented reality or image recognition, users expect quick responses despite potential minor decreases in accuracy.

Critical decision-making systems, such as medical diagnosis or security screening, prioritize accuracy due to the significant consequences associated with object detection errors. However, maintaining high inference speeds remains crucial for timely diagnoses or responses. Striking a balance between accuracy and speed becomes paramount in such applications to uphold reliability without sacrificing performance.

In large-scale surveillance systems tasked with monitoring vast areas or crowds, accurate object detection is essential for identifying potential threats. Yet, processing extensive amounts of video data in real-time necessitates high inference speeds to promptly detect and respond to suspicious activities. Tailoring trade-offs between accuracy and speed to suit the specific requirements of the surveillance environment is therefore vital.

Even within the realm of machine learning research and development, where experimentation and model iteration are frequent, there exists a trade-off between training time and model accuracy. Researchers often prioritize faster inference speeds during prototyping and experimentation stages to iterate rapidly on model architectures and hyperparameters. Subsequently, more intensive training with a focus on achieving higher accuracy may be pursued once promising models are identified. Ultimately, finding the appropriate balance between accuracy and inference speed is crucial to ensuring optimal performance across diverse use cases and applications.

VI. CONCLUSION

In this study, we discuss object detection powered by deep learning. Leveraging the training capabilities of Convolutional Neural Networks (CNNs), we can effectively identify thousands of objects across diverse scenarios, including challenging lighting conditions and occlusions. The sheer volume of training images available supports CNNs, enabling them to achieve unmatched accuracy across a multitude of classes.

Modern object detectors that utilize CNNs excel in pinpointing an object's position within an image and subsequently classifying it. We examine the architectures of several leading object detection frameworks, highlighting their unique contributions and how they differentiate from other models. Our evaluation is based on tests conducted using the COCO test-dev2017 dataset, where we compare their performance in precision and inference speed.

This study provides an overview of deep learning-driven object detectors, guiding computer vision practitioners in choosing the appropriate tool for various applications. Moreover, by contrasting the different architectures and their results, we aim to inspire improvements in current designs or even catalyze the creation of innovative models.

REFERENCES

[1] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Zürich, Switzerland: Springer, Sep. 2014, pp. 740–755.

[2] N. Marku, M. Frljak, I. S. Pandzic, J. Ahlberg, and R. Forchheimer, "A method for object detection based on pixel intensity comparisons," 2013, *arXiv:1305.4537*.

[3] I. Kalinovskii and V. Spitsyn, "Compact convolutional neural network cascade for face detection," in *Proc. Parallel Comput. Technol. (PCT)*, Arkhangelsk, Russia, Mar./Apr. 2016.

[4] V. Kastrinaki, M. Zervakis, and K. Kalaitzakis, "A survey of video processing techniques for traffic applications," *Image Vis. Comput.*, vol. 21, no. 4, pp. 359–381, Apr. 2003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0262885603000040

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, Oct. 1999, pp. 1150–1157.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[7] T. Lindeberg, "Scale invariant feature transform," *Scholarpedia*, vol. 7, p. 10491, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:5900376

[8] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. 9th Eur. Conf. Comput. Vis. (ECCV)*. Graz, Austria: Springer, May 2006, pp. 430–443.

[9] L. Christodoulou, T. Kasparis, and O. Marques, "Advanced statistical and adaptive threshold techniques for moving object detection and segmentation," in *Proc. 17th Int. Conf. Digit. Signal Process. (DSP)*, Jul. 2011, pp. 1–6.

[10] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Proc. 9th Eur. Conf. Comput. Vis.*, vol. 3951, 2006, pp. 404–417.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 886–893.

[12] M. M. El-gayar, H. Soliman, and N. Meky, "A comparative study of image low level feature extraction algorithms," *Egyptian Informat. J.*, vol. 14, no. 2, pp. 175–181, Jul. 2013.

[13] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*. Springer, 1982, pp. 267–285.

[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[15] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2, 1989, pp. 1–9.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[20] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *Int. J. Comput. Vis.*, vol. 128, pp. 261–318, Feb. 2019.

[21] K. L. Masita, A. N. Hasan, and T. Shongwe, "Deep learning in object detection: A review," in *Proc. Int. Conf. Artif. Intell., Big Data, Comput. Data Commun. Syst. (icABCD)*, Aug. 2020, pp. 1–11.

[22] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.

[23] X. Wu, D. Sahoo, and S. C. H. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, Jul. 2020.

[24] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019, *arXiv:1905.05055*.

[25] V. K. Sharma and R. N. Mir, "A comprehensive and systematic look up into deep learning based object detection techniques: A review," *Comput. Sci. Rev.*, vol. 38, Nov. 2020, Art. no. 100301.

[26] L. Aziz, Md. S. B. Haji Salam, U. U. Sheikh, and S. Ayub, "Exploring deep learning-based architecture, strategies, applications and current trends in generic object detection: A comprehensive review," *IEEE Access*, vol. 8, pp. 170461–170495, 2020.

[27] S. Agarwal, J. Ogier Du Terrail, and F. Jurie, "Recent advances in object detection in the age of deep convolutional neural networks," 2018, *arXiv:1809.03193*.

[28] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan, "A review of object detection based on deep learning," *Multimedia Tools Appl.*, vol. 79, no. 33, pp. 23729–23791, 2020.

[29] H. Zhang and X. Hong, "Recent progresses on object detection: A brief review," *Multimedia Tools Appl.*, vol. 78, no. 19, pp. 27809–27847, Oct. 2019.

[30] S. Liu, H. Zhou, C. Li, and S. Wang, "Analysis of anchor-based and anchor-free object detection methods based on deep learning," in *Proc. IEEE Int. Conf. Mechatronics Autom. (ICMA)*, Oct. 2020, pp. 1058–1065.

[31] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation (R-CNN)," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[32] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[35] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7263–7271.

[36] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[37] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[38] C.-Y. Wang, A. Bochkovskiy, and H. M. Liao, "Scaled-YOLOv4: Scaling cross stage partial network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13024–13033.

[39] C.-Y. Wang, I.-H. Yeh, and H.-Y. Mark Liao, "You only learn one representation: Unified network for multiple tasks," 2021, *arXiv:2105.04206*.

[40] G. Jocher. (May 2020). *YOLOv5 By Ultralytics*. [Online]. Available: https://github.com/ultralytics/yolov5

[41] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen, "PP-YOLO: An effective and efficient implementation of object detector," 2020, *arXiv:2007.12099*.

[42] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, D. Yu, Y. Ma, and O. Yoshie, "PP-YOLOv2: A practical object detector," 2021, *arXiv:2104.10419*.

[43] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.

[44] S. Xu, X. Wang, W. Lv, Q. Chang, C. Cui, K. Deng, G. Wang, Q. Dang, S. Wei, Y. Du, and B. Lai, "PP-YOLOE: An evolved version of YOLO," 2022, *arXiv:2203.16250*.

[45] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "YOLOv6: A single-stage object detection framework for industrial applications," 2022, *arXiv:2209.02976*.

[46] C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 7464–7475.

[47] X. Xu, Y. Jiang, W. Chen, Y. Huang, Y. Zhang, and X. Sun, "DAMO-YOLO: A report on real-time object detection design," 2022, *arXiv:2211.15444*.

[48] G. Jocher, A. Chaurasia, and J. Qiu. (Jan. 2023). *Ultralytics YOLO*. [Online]. Available: https://github.com/ultralytics/ultralytics

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* (Lecture Notes in Computer Science), vol. 9905, Oct. 2016, pp. 21–37.

[50] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[51] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2008, pp. 1–8.

[52] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2241–2248.

[53] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," 2019, *arXiv:1911.09070*.

[54] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation (FCNs)," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.

[55] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection (RetinaNet)," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, Feb. 2017.

[56] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9626–9635.

[57] K. Oksuz, B. Can Cam, S. Kalkan, and E. Akbas, "Imbalance problems in object detection: A review," 2019, *arXiv:1909.00169*.

[58] H. Law and J. Deng, "CornerNet: Detecting objects as paired keypoints," *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 642–656, Mar. 2020.

[59] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, Feb. 2020.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition (VGG Net)," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[61] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
[62] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, Jun. 2006, pp. 2169–2178.
[63] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 558–567.
[64] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 764–773.
[65] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection (FPNs)," in *Proc. CVPR*, vol. 3, no. 3, 2017, pp. 137–147.
[66] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8759–8768.
[67] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. 36th Int. Conf. Mach. Learn.*, Jun. 2019, pp. 10691–10700.
[68] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," 2014, *arXiv:1312.6229*.
[69] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[70] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2020, pp. 213–229.
[71] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," 2021, *arXiv:2102.12122*.
[72] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "PVTv2: Improved baselines with pyramid vision transformer," 2021, *arXiv:2106.13797*.
[73] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.
[74] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021, *arXiv:2103.14030*.

KHALID ELGAZZAR (Senior Member, IEEE) received the B.Sc. degree in computer and communication engineering from Alexandria University, Egypt, the M.Sc. degree in computer engineering from Arab Academy for Science and Technology, Egypt, and the Ph.D. degree in computer science from Queens University, Canada. He is currently the Canada Research Chair of the Internet of Things (IoT) and an Associate Professor with the Department of Electrical, Computer, and Software Engineering, Ontario Tech University, where he is also the Founder and the Director of the IoT Research Laboratory. Before joining Ontario Tech University, he was an Assistant Professor with the University of Louisiana at Lafayette and a Research Associate with the Carnegie Mellon School of Computer Science. He is an expert in the areas of the Internet of Things (IoT), artificial intelligence, computer systems, real-time data analytics, and mobile computing. He is an active volunteer in technical program committees and organizing committees in both IEEE and ACM events. He received many research awards and several recognitions and best paper awards at top international venues. He is an associate editor for several ACM/IEEE journals in the areas of mobile computing and the IoT.

SIFATUL MOSTAFI (Member, IEEE) received the B.Sc. degree in computer science and engineering (economics) from BRAC University, Bangladesh, in 2018, and the M.A.Sc. degree in software systems engineering from Ontario Tech University, Canada, in 2021. He is currently pursuing the Ph.D. degree in software systems engineering with the IoT Research Laboratory, Ontario Tech University. He is a Doctoral Researcher with the IoT Research Laboratory, Ontario Tech University. His research interests include improving traffic safety and efficiency using real-time predictive traffic modeling and simulation. He serves several scientific communities, including the IEEE Intelligent Transportation Systems Society (ITSS) in numerous aspects.

REED DENNIS received the B.Sc. degree in computer systems and software engineering from Trent University, Canada, in 2018, and the M.Sc. degree in software engineering from Ontario Tech University, Canada, in 2022. He is an AI Researcher. His research interests include computer vision applications for intelligent transportation systems, deep learning-based object detection, applied machine learning, and metaheuristics.

YOUSSEF OSMAN received the B.Sc. degree in computer science from Ain Shams University, Egypt, and the M.Sc. degree in software engineering from Ontario Tech University, Canada. He is currently a Data Engineer with the TD Bank Group. During his master's studies, he has made contributions to the field of deep learning and computer vision applied in precision agriculture. His current work involves big data analysis and engineering, working with enterprise bank data in the cloud. His goal is to broaden his knowledge base in order to work in data architecture, which involves designing complete end-to-end data solutions starting from ingesting source data, to manipulating that data, to analyzing and visualizing the data. His research interests include data science, data engineering, and analysis.