A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 AND BEYOND

UNDER REVIEW IN ACM COMPUTING SURVEYS

Juan R. Terven
CICATA-Qro
Instituto Politecnico Nacional
Mexico
jrtervens@ipn.mx

Diana M. Cordova-Esparaza
Facultad de Informática
Universidad Autónoma de Querétaro
Mexico
diana.cordova@uaq.mx

May 22, 2023

ABSTRACT

YOLO has become a central real-time object detection system for robotics, driverless cars, and video monitoring applications. We present a comprehensive analysis of YOLO's evolution, examining the innovations and contributions in each iteration from the original YOLO to YOLOv8 and YOLO-NAS. We start by describing the standard metrics and postprocessing; then, we discuss the major changes in network architecture and training tricks for each model. Finally, we summarize the essential lessons from YOLO's development and provide a perspective on its future, highlighting potential research directions to enhance real-time object detection systems.

Keywords YOLO · Object detection · Deep Learning · Computer Vision

1 Introduction

Real-time object detection has emerged as a critical component in numerous applications, spanning various fields such as autonomous vehicles, robotics, video surveillance, and augmented reality. Among the various object detection algorithms, the YOLO (You Only Look Once) framework has stood out for its remarkable balance of speed and accuracy, enabling the rapid and reliable identification of objects in images. Since its inception, the YOLO family has evolved through multiple iterations, each building upon the previous versions to address limitations and enhance performance (see Figure 1). This paper aims to provide a comprehensive review of the YOLO framework's development, from the original YOLOv1 to the latest YOLOv8, elucidating the key innovations, differences, and improvements across each version.

The paper begins by exploring the foundational concepts and architecture of the original YOLO model, which set the stage for the subsequent advances in the YOLO family. Following this, we delve into the refinements and enhancements introduced in each version, ranging from YOLOv2 to YOLOv8. These improvements encompass various aspects such as network design, loss function modifications, anchor box adaptations, and input resolution scaling. By examining these developments, we aim to offer a holistic understanding of the YOLO framework's evolution and its implications for object detection.

In addition to discussing the specific advancements of each YOLO version, the paper highlights the trade-offs between speed and accuracy that have emerged throughout the framework's development. This underscores the importance of considering the context and requirements of specific applications when selecting the most appropriate YOLO model. Finally, we envision the future directions of the YOLO framework, touching upon potential avenues for further research and development that will shape the ongoing progress of real-time object detection systems.

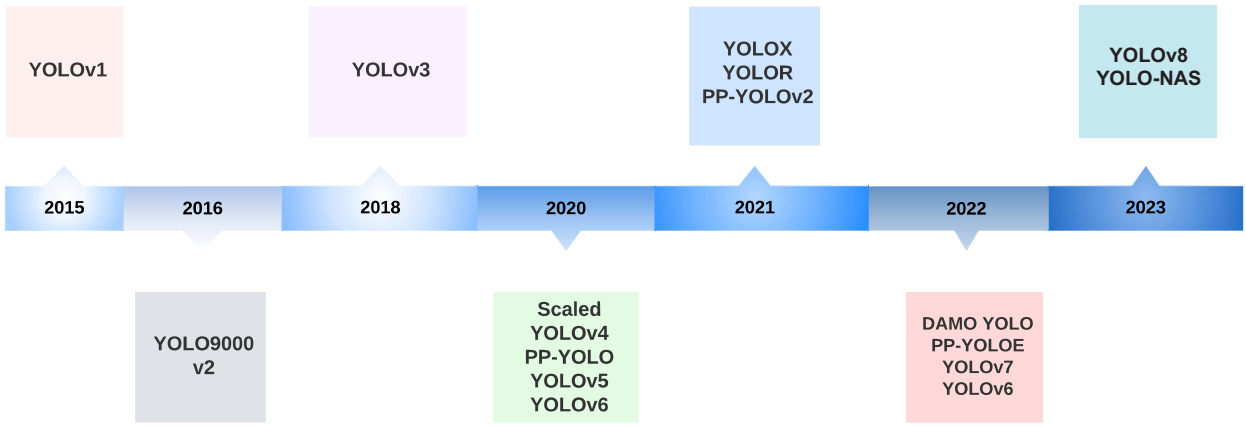

Figure 1: A timeline of YOLO versions.

2 YOLO Applications Across Diverse Fields

YOLO's real-time object detection capabilities have been invaluable in autonomous vehicle systems, enabling quick identification and tracking of various objects such as vehicles, pedestrians [1, 2], bicycles, and other obstacles [3, 4, 5, 6]. These capabilities have been applied in numerous fields, including action recognition [7] in video sequences for surveillance [8], sports analysis [9], and human-computer interaction [10].

YOLO models have been used in agriculture to detect and classify crops [11, 12], pests, and diseases [13], assisting in precision agriculture techniques and automating farming processes. They have also been adapted for face detection tasks in biometrics, security, and facial recognition systems [14, 15].

In the medical field, YOLO has been employed for cancer detection [16, 17], skin segmentation [18], and pill identification [19], leading to improved diagnostic accuracy and more efficient treatment processes. In remote sensing, it has been used for object detection and classification in satellite and aerial imagery, aiding in land use mapping, urban planning, and environmental monitoring [20, 21, 22, 23].

Security systems have integrated YOLO models for real-time monitoring and analysis of video feeds, allowing rapid detection of suspicious activities [24], social distancing, and face mask detection [25]. The models have also been applied in surface inspection to detect defects and anomalies, enhancing quality control in manufacturing and production processes [26, 27, 28].

In traffic applications, YOLO models have been utilized for tasks such as license plate detection [29] and traffic sign recognition [30], contributing to the development of intelligent transportation systems and traffic management solutions. They have been employed in wildlife detection and monitoring to identify endangered species for biodiversity conservation and ecosystem management [31]. Lastly, YOLO has been widely used in robotic applications [32, 33] and object detection from drones [34, 35].

3 Object Detection Metrics and Non-Maximum Suppression (NMS)

The Average Precision (AP), traditionally called *Mean Average Precision* (mAP), is the commonly used metric for evaluating the performance of object detection models. It measures the average precision across all categories, providing a single value to compare different models. The COCO dataset makes no distinction between AP and mAP. In the rest of this paper, we will refer to this metric as AP.

In YOLOv1 and YOLOv2, the dataset utilized for training and benchmarking was PASCAL VOC 2007, and VOC 2012 [36]. However, from YOLOv3 onwards, the dataset used is Microsoft COCO (Common Objects in Context) [37]. The AP is calculated differently for these datasets. The following sections will discuss the rationale behind AP and explain how it is computed.

3.1 How AP works?

The AP metric is based on precision-recall metrics, handling multiple object categories, and defining a positive prediction using Intersection over Union (IoU).

Precision and Recall: Precision measures the accuracy of the model's positive predictions, while recall measures the proportion of actual positive cases that the model correctly identifies. There is often a trade-off between precision and recall; for example, increasing the number of detected objects (higher recall) can result in more false positives (lower precision). To account for this trade-off, the AP metric incorporates the precision-recall curve that plots precision against recall for different confidence thresholds. This metric provides a balanced assessment of precision and recall by considering the area under the precision-recall curve.

Handling multiple object categories: Object detection models must identify and localize multiple object categories in an image. The AP metric addresses this by calculating each category's average precision (AP) separately and then taking the mean of these APs across all categories (that is why it is also called mean average precision). This approach ensures that the model's performance is evaluated for each category individually, providing a more comprehensive assessment of the model's overall performance.

Intersection over Union: Object detection aims to accurately localize objects in images by predicting bounding boxes. The AP metric incorporates the Intersection over Union (IoU) measure to assess the quality of the predicted bounding boxes. IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box (see Figure 2). It measures the overlap between the ground truth and predicted bounding boxes. The COCO benchmark considers multiple IoU thresholds to evaluate the model's performance at different levels of localization accuracy.

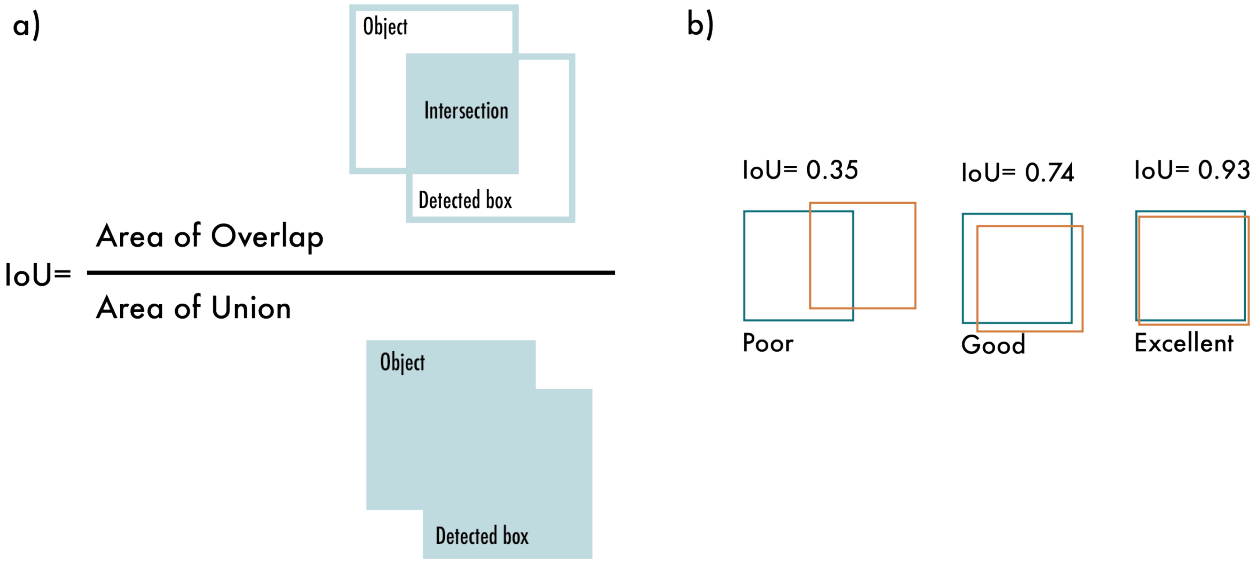

Figure 2: Intersection over Union (IoU). a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; b) examples of three different IoU values for different box locations.

3.2 Computing AP

The AP is computed differently in the VOC and in the COCO datasets. In this section, we describe how it is computed on each dataset.

VOC Dataset

This dataset includes 20 object categories. To compute the AP in VOC, we follow the next steps:

1. For each category, calculate the precision-recall curve by varying the confidence threshold of the model's predictions.

2. Calculate each category's average precision (AP) using an interpolated 11-point sampling of the precision-recall curve.
3. Compute the final average precision (AP) by taking the mean of the APs across all 20 categories.

Microsoft COCO Dataset

This dataset includes 80 object categories and uses a more complex method for calculating AP. Instead of using an 11-point interpolation, it uses a 101-point interpolation, i.e., it computes the precision for 101 recall thresholds from 0 to 1 in increments of 0.01. Also, the AP is obtained by averaging over multiple IoU values instead of just one, except for a common AP metric called AP_{50}, which is the AP for a single IoU threshold of 0.5. The steps for computing AP in COCO are the following:

1. For each category, calculate the precision-recall curve by varying the confidence threshold of the model's predictions.
2. Compute each category's average precision (AP) using 101-recall thresholds.
3. Calculate AP at different Intersection over Union (IoU) thresholds, typically from 0.5 to 0.95 with a step size of 0.05. A higher IoU threshold requires a more accurate prediction to be considered a true positive.
4. For each IoU threshold, take the mean of the APs across all 80 categories.
5. Finally, compute the overall AP by averaging the AP values calculated at each IoU threshold.

The differences in AP calculation make it hard to directly compare the performance of object detection models across the two datasets. The current standard uses the COCO AP due to its more fine-grained evaluation of how well a model performs at different IoU thresholds.

3.3 Non-Maximum Suppression (NMS)

Non-Maximum Suppression (NMS) is a post-processing technique used in object detection algorithms to reduce the number of overlapping bounding boxes and improve the overall detection quality. Object detection algorithms typically generate multiple bounding boxes around the same object with different confidence scores. NMS filters out redundant and irrelevant bounding boxes, keeping only the most accurate ones. Algorithm 1 describes the procedure. Figure 3 shows the typical output of an object detection model containing multiple overlapping bounding boxes and the output after NMS.

Algorithm 1 Non-Maximum Suppression Algorithm

Require: Set of predicted bounding boxes B, confidence scores S, IoU threshold τ, confidence threshold T
Ensure: Set of filtered bounding boxes F

$F \leftarrow \emptyset$
Filter the boxes: $B \leftarrow \{b \in B \mid S(b) \geq T\}$
Sort the boxes B by their confidence scores in descending order
while $B \neq \emptyset$ **do**
    Select the box b with the highest confidence score
    Add b to the set of final boxes F: $F \leftarrow F \cup \{b\}$
    Remove b from the set of boxes B: $B \leftarrow B - \{b\}$
    for all remaining boxes r in B **do**
        Calculate the IoU between b and r: $iou \leftarrow IoU(b, r)$
        if $iou \geq \tau$ **then**
            Remove r from the set of boxes B: $B \leftarrow B - \{r\}$
        end if
    end for
end while

We are ready to start describing the different YOLO models.

4 YOLO: You Only Look Once

YOLO by Joseph Redmon et al. was published in CVPR 2016 [38]. It presented for the first time a real-time end-to-end approach for object detection. The name YOLO stands for "You Only Look Once," referring to the fact that it was

Figure 3: Non-Maximum Suppression (NMS). a) Shows the typical output of an object detection model containing multiple overlapping boxes. b) Shows the output after NMS.

able to accomplish the detection task with a single pass of the network, as opposed to previous approaches that either used sliding windows followed by a classifier that needed to run hundreds or thousands of times per image or the more advanced methods that divided the task into two-steps, where the first step detects possible regions with objects or *regions proposals* and the second step run a classifier on the proposals. Also, YOLO used a more straightforward output based only on regression to predict the detection outputs as opposed to Fast R-CNN [39] that used two separate outputs, a classification for the probabilities and a regression for the boxes coordinates.

4.1 How YOLOv1 works?

YOLOv1 unified the object detection steps by detecting all the bounding boxes simultaneously. To accomplish this, YOLO divides the input image into a $S \times S$ grid and predicts B bounding boxes of the same class, along with its confidence for C different classes per grid element. Each bounding box prediction consists of five values: Pc, bx, by, bh, bw where Pc is the confidence score for the box that reflects how confident the model is that the box contains an object and how accurate the box is. The bx and by coordinates are the centers of the box relative to the grid cell, and bh and bw are the height and width of the box relative to the full image. The output of YOLO is a tensor of $S \times S \times (B \times 5 + C)$ optionally followed by non-maximum suppression (NMS) to remove duplicate detections.

In the original YOLO paper, the authors used the PASCAL VOC dataset [36] that contains 20 classes ($C = 20$); a grid of 7×7 ($S = 7$) and at most 2 classes per grid element ($B = 2$), giving a $7 \times 7 \times 30$ output prediction.

Figure 4 shows a simplified output vector considering a three-by-three grid, three classes, and a single class per grid for eight values. In this simplified case, the output of YOLO would be $3 \times 3 \times 8$.

YOLOv1 achieved an average precision (AP) of 63.4 on the PASCAL VOC2007 dataset.

4.2 YOLOv1 Architecture

YOLOv1 architecture comprises 24 convolutional layers followed by two fully-connected layers that predict the bounding box coordinates and probabilities. All layers used leaky rectified linear unit activations [40] except for the last one that used a linear activation function. Inspired by GoogLeNet [41] and Network in Network [42], YOLO uses 1×1 convolutional layers to reduce the number of feature maps and keep the number of parameters relatively low. As activation layers, Table 1 describes the YOLOv1 architecture. The authors also introduced a lighter model called Fast YOLO, composed of nine convolutional layers.

4.3 YOLOv1 Training

The authors pre-trained the first 20 layers of YOLO at a resolution of 224×224 using the ImageNet dataset [43]. Then, they added the last four layers with randomly initialized weights and fine-tuned the model with the PASCAL VOC 2007, and VOC 2012 datasets [36] at a resolution of 448×448 to increase the details for more accurate object detection.

For augmentations, the authors used random scaling and translations of at most 20% of the input image size, as well as random exposure and saturation with an upper-end factor of 1.5 in the HSV color space.

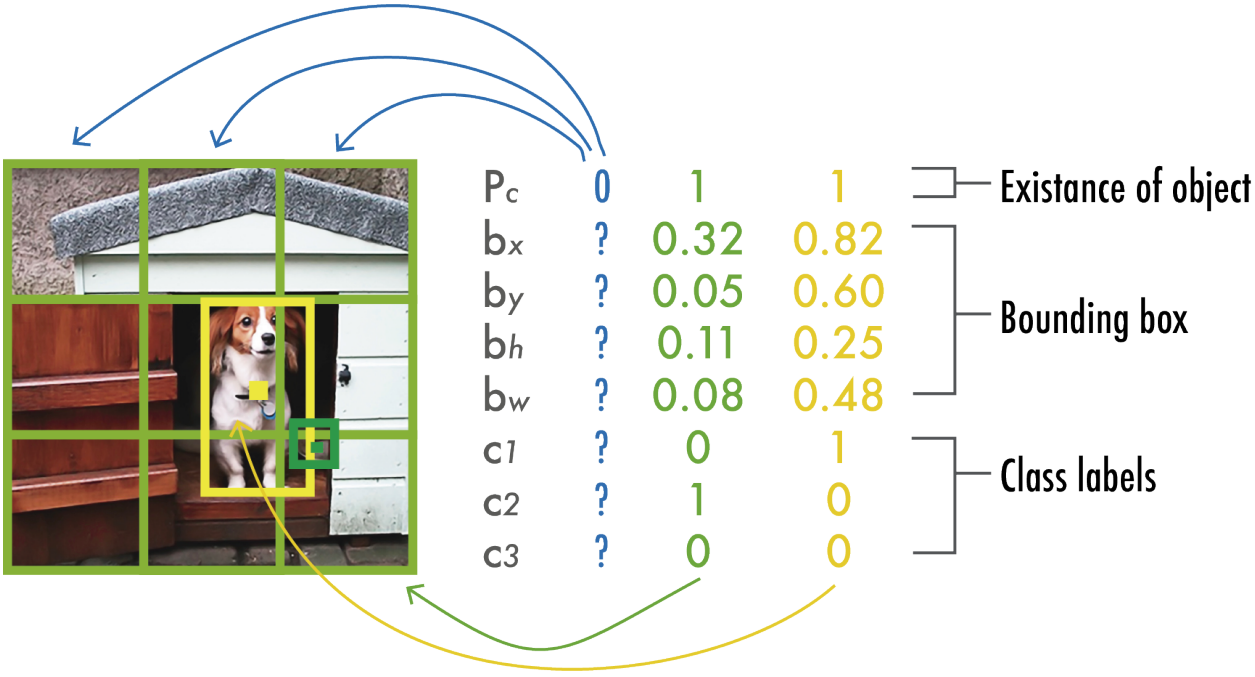

Figure 4: YOLO output prediction. The figure depicts a simplified YOLO model with a three-by-three grid, three classes, and a single class prediction per grid element to produce a vector of eight values.

YOLOv1 used a loss function composed of multiple sum-squared errors, as shown in Figure 5. In the loss function, $\lambda_{coord} = 5$ is a scale factor that gives more importance to the bounding boxes predictions, and $\lambda_{noobj} = 0.5$ is a scale factor that decreases the importance of the boxes that do not contain objects.

The first two terms of the loss represent the *localization loss*; it computes the error in the predicted bounding boxes locations (x, y) and sizes (w, h). Note that these errors are only computed in the boxes containing objects (represented by the $\mathbb{1}_{ij}^{obj}$), only penalizing if an object is present in that grid cell. The third and fourth loss terms represent the *confidence loss*; the third term measures the confidence error when the object is detected in the box ($\mathbb{1}_{ij}^{obj}$) and the fourth term measures the confidence error when the object is not detected in the box ($\mathbb{1}_{ij}^{noobj}$). Since most boxes are empty, this loss is weighted down by the λ_{noobj} term. The final loss component is the *classification loss* that measures the squared error of the class conditional probabilities for each class only if the object appears in the cell ($\mathbb{1}_{i}^{obj}$).

4.4 YOLOv1 Strengths and Limitations

The simple architecture of YOLO, along with its novel full-image one-shot regression, made it much faster than the existing object detectors allowing real-time performance.

However, while YOLO performed faster than any object detector, the localization error was larger compared with state-of-the-art methods such as Fast R-CNN [39]. There were three major causes of this limitation:

1. It could only detect at most two objects of the same class in the grid cell, limiting its ability to predict nearby objects.

2. It struggled to predict objects with aspect ratios not seen in the training data.

3. It learned from coarse object features due to the down-sampling layers.

Table 1: YOLO Architecture. The architecture comprises 24 convolutional layers combining 3×3 convolutions with 1×1 convolutions for channel reduction. The output is a fully connected layer that generates a grid of 7×7 with 30 values for each grid cell to accommodate ten bounding box coordinates (2 boxes) with 20 categories.

	Type	Filters	Size/Stride	Output
	Conv	64	7×7 / 2	224×224
	Max Pool		2×2 / 2	112×112
	Conv	192	3×3 / 1	112×112
	Max Pool		2×2 / 2	56×56
$1\times$	Conv	128	1×1 / 1	56×56
	Conv	256	3×3 / 1	56×56
	Conv	256	1×1 / 1	56×56
	Conv	512	3×3 / 1	56×56
	Max Pool		2×2 / 2	28×28
$4\times$	Conv	256	1×1 / 1	28×28
	Conv	512	3×3 / 1	28×28
	Conv	512	1×1 / 1	28×28
	Conv	1024	3×3 / 1	28×28
	Max Pool		2×2 / 2	14×14
$2\times$	Conv	512	1×1 / 1	14×14
	Conv	1024	3×3 / 1	14×14
	Conv	1024	3×3 / 1	14×14
	Conv	1024	3×3 / 2	7×7
	Conv	1024	3×3 / 1	7×7
	Conv	1024	3×3 / 1	7×7
	FC		4096	4096
	Dropout 0.5			4096
	FC		$7 \times 7 \times 30$	$7 \times 7 \times 30$

5 YOLOv2: Better, Faster, and Stronger

YOLOv2 was published in CVPR 2017 [44] by Joseph Redmon and Ali Farhadi. It included several improvements over the original YOLO, to make it better, keeping the same speed and also stronger —capable of detecting 9000 categories!—. The improvements were the following:

1. **Batch normalization** on all convolutional layers improved convergence and acts as a regularizer to reduce overfitting.
2. **High-resolution classifier**. Like YOLOv1, they pre-trained the model with ImageNet at 224×224. However, this time, they finetuned the model for ten epochs on ImageNet with a resolution of 448×448, improving the network performance on higher resolution input.
3. **Fully convolutional**. They removed the dense layers and used a fully convolutional architecture.
4. **Use anchor boxes to predict bounding boxes**. They use a set of *prior boxes* or *anchor boxes*, which are boxes with predefined shapes used to match prototypical shapes of objects as shown in Figure 6. Multiple anchor boxes are defined for each grid cell, and the system predicts the coordinates and the class for every anchor box. The size of the network output is proportional to the number of anchor boxes per grid cell.
5. **Dimension Clusters**. Picking good prior boxes helps the network learn to predict more accurate bounding boxes. The authors ran k-means clustering on the training bounding boxes to find good priors. They selected five prior boxes providing a good tradeoff between recall and model complexity.
6. **Direct location prediction**. Unlike other methods that predicted offsets [45], YOLOv2 followed the same philosophy and predicted location coordinates relative to the grid cell. The network predicts five bounding boxes for each cell, each with five values t_x, t_y, t_w, t_h, and t_o, where t_o is equivalent to Pc from YOLOv1 and the final bounding box coordinates are obtained as shown in Figure 7.
7. **Finner-grained features**. YOLOv2, compared with YOLOv1, removed one pooling layer to obtain an output feature map or grid of 13×13 for input images of 416×416. YOLOv2 also uses a passthrough layer that takes the $26 \times 26 \times 512$ feature map and reorganizes it by stacking adjacent features into different channels

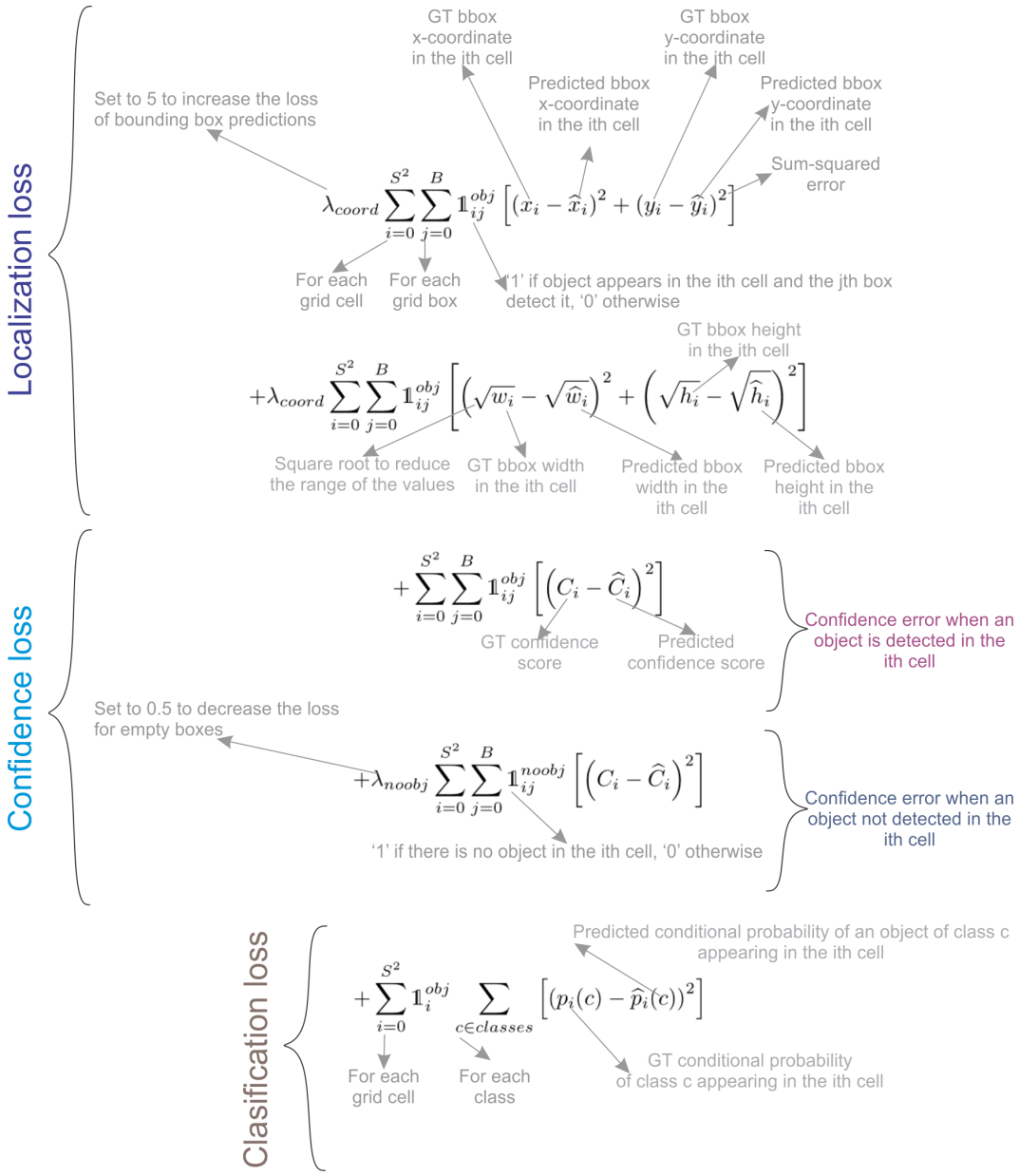

Figure 5: YOLO cost function: includes localization loss for bounding box coordinates, confidence loss for object presence or absence, and classification loss for category prediction accuracy.

instead of losing them via a spatial subsampling. This generates $13 \times 13 \times 2048$ feature maps concatenated in the channel dimension with the lower resolution $13 \times 13 \times 1024$ maps to obtain $13 \times 13 \times 3072$ feature maps. See Table 2 for the architectural details.

8. **Multi-scale training**. Since YOLOv2 does not use fully connected layers, the inputs can be different sizes. To make YOLOv2 robust to different input sizes, the authors trained the model randomly, changing the input size —from 320×320 up to 608×608— every ten batches.

With all these improvements, YOLOv2 achieved an average precision (AP) of 78.6% on the PASCAL VOC2007 dataset compared to the 63.4% obtained by YOLOv1.

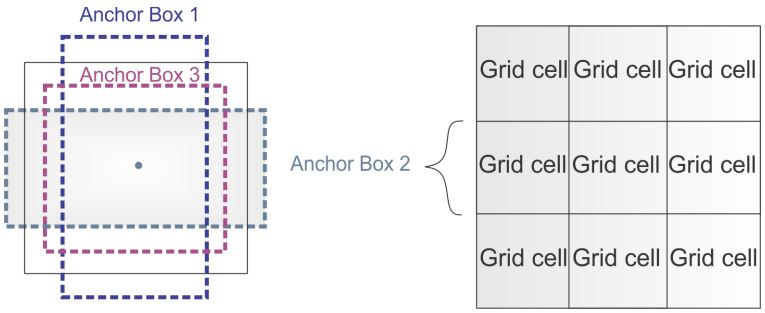

Figure 6: Anchor boxes. YOLOv2 defines multiple anchor boxes for each grid cell.

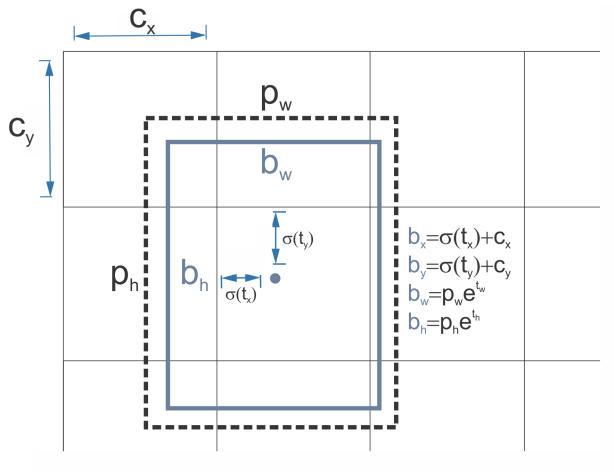

Figure 7: Bounding boxes prediction. The box's center coordinates are obtained with the predicted t_x, t_y values passing through a sigmoid function and offset by the location of the grid cell c_x, c_y. The width and height of the final box use the prior width p_w and height p_h scaled by e^{t_w} and e^{t_h} respectively, where t_w and t_h are predicted by YOLOv2.

5.1 YOLOv2 Architecture

The backbone architecture used by YOLOv2 is called *Darknet-19*, containing 19 convolutional layers and five max-pooling layers. Similar to the architecture of YOLOv1, it is inspired in the Network in Network [42] using 1×1 convolutions between the 3×3 to reduce the number of parameters. In addition, as mentioned above, they use batch normalization to regularize and help convergence.

Table 2 shows the entire Darknet-19 backbone with the object detection head. YOLOv2 predicts five bounding boxes, each with five values and 20 classes when using the PASCAL VOC dataset.

The object classification head replaces the last four convolutional layers with a single convolutional layer with 1000 filters, followed by a global average pooling layer and a Softmax.

5.2 YOLO9000 is a stronger YOLOv2

The authors introduced a method for training joint classification and detection in the same paper. It used the detection labeled data from COCO [37] to learn bounding box coordinates and classification data from ImageNet to increase the number of categories it can detect. During training, they combined both datasets such that when a detection training image is used, it backpropagates the detection network, and when a classification training image is used, it backpropagates the classification part of the architecture. The result is a YOLO model capable of detecting more than 9000 categories hence the name YOLO9000.

Table 2: YOLOv2 Architecture. Darknet-19 backbone (layers 1 to 23) plus the detection head composed of the last four convolutional layers and the passthrough layer that reorganizes the features of the 17th output of $26 \times 26 \times 512$ into $13 \times 13 \times 2048$ followed by concatenation with the 25th layer. The final convolution generates a grid of 13×13 with 125 channels to accommodate 25 predictions (5 coordinates + 20 classes) for five bounding boxes.

Num	Type	Filters	Size/Stride	Output
1	Conv/BN	32	3×3 / 1	$416 \times 416 \times 32$
2	Max Pool		2×2 / 2	$208 \times 208 \times 32$
3	Conv/BN	64	3×3 / 1	$208 \times 208 \times 64$
4	Max Pool		2×2 / 2	$104 \times 104 \times 64$
5	Conv/BN	128	3×3 / 1	$104 \times 104 \times 128$
6	Conv/BN	64	1×1 / 1	$104 \times 104 \times 64$
7	Conv/BN	128	3×3 / 1	$104 \times 104 \times 128$
8	Max Pool		2×2 / 2	$52 \times 52 \times 128$
9	Conv/BN	256	3×3 / 1	$52 \times 52 \times 256$
10	Conv/BN	128	1×1 / 1	$52 \times 52 \times 128$
11	Conv/BN	256	3×3 / 1	$52 \times 52 \times 256$
12	Max Pool		2×2 / 2	$52 \times 52 \times 256$
13	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
14	Conv/BN	256	1×1 / 1	$26 \times 26 \times 256$
15	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
16	Conv/BN	256	1×1 / 1	$26 \times 26 \times 256$
17	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
18	Max Pool		2×2 / 2	$13 \times 13 \times 512$
19	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
20	Conv/BN	512	1×1 / 1	$13 \times 13 \times 512$
21	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
22	Conv/BN	512	1×1 / 1	$13 \times 13 \times 512$
23	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
24	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
25	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
26	Reorg layer 17			$13 \times 13 \times 2048$
27	Concat 25 and 26			$13 \times 13 \times 3072$
28	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
29	Conv	125	1×1 / 1	$13 \times 13 \times 125$

6 YOLOv3

YOLOv3 [46] was published in ArXiv in 2018 by Joseph Redmon and Ali Farhadi. It included significant changes and a bigger architecture to be on par with the state-of-the-art while keeping real-time performance. In the following, we described the changes with respect to YOLOv2.

1. **Bounding box prediction**. Like YOLOv2, the network predicts four coordinates for each bounding box t_x, t_y, t_w, and t_h; however, this time, YOLOv3 predicts an *objectness score* for each bounding box using logistic regression. This score is 1 for the anchor box with the highest overlap with the ground truth and 0 for the rest anchor boxes. YOLOv3, as opposed to Faster R-CNN [45], assigns only one anchor box to each ground truth object. Also, if no anchor box is assigned to an object, it only incurs in classification loss but not localization loss or confidence loss.
2. **Class Prediction**. Instead of using a softmax for the classification, they used binary cross-entropy to train independent logistic classifiers and pose the problem as a multilabel classification. This change allows assigning multiple labels to the same box, which may occur on some complex datasets [47] with overlapping labels. For example, the same object can be a *Person* and a *Man*.
3. **New backbone**. YOLOv3 features a larger feature extractor composed of 53 convolutional layers with residual connections. Section 6.1 describes the architecture in more detail.
4. **Spatial pyramid pooling (SPP)** Although not mentioned in the paper, the authors also added to the backbone a modified SPP block [48] that concatenates multiple max pooling outputs without subsampling (stride = 1), each with different kernel sizes $k \times k$ where $k = 1, 5, 9, 13$ allowing a larger receptive field. This version is called YOLOv3-spp and was the best-performed version improving the AP_{50} by 2.7%.

5. **Multi-scale Predictions**. Similar to Feature Pyramid Networks [49], YOLOv3 predicts three boxes at three different scales. Section 6.2 describes the multi-scale prediction mechanism with more details.
6. **Bounding box priors**. Like YOLOv2, the authors also use k-means to determine the bounding box priors of anchor boxes. The difference is that in YOLOv2, they used a total of five prior boxes per cell, and in YOLOv3, they used three prior boxes for three different scales.

6.1 YOLOv3 Architecture

The architecture backbone presented in YOLOv3 is called Darknet-53. It replaced all max-pooling layers with strided convolutions and added residual connections. In total, it contains 53 convolutional layers. Figure 8 shows the architecture details.

The Darknet-53 backbone obtains Top-1 and Top-5 accuracies comparable with ResNet-152 but almost $2\times$ faster.

Layer	Filters	size	Repeat		Output size
Image					416×416
Conv	32	$3 \times 3/1$	1		416×416
Conv	64	$3 \times 3/2$	1		208×208
Conv	32	$1 \times 1/1$	Conv		208×208
Conv	64	$3 \times 3/1$	Conv	$\times 1$	208×208
Residual			Residual		208×208
Conv	128	$3 \times 3/2$	1		104×104
Conv	64	$1 \times 1/1$	Conv		104×104
Conv	128	$3 \times 3/1$	Conv	$\times 2$	104×104
Residual			Residual		104×104
Conv	256	$3 \times 3/2$	1		52×52
Conv	128	$1 \times 1/1$	Conv		52×52
Conv	256	$3 \times 3/1$	Conv	$\times 8$	52×52
Residual			Residual		52×52
Conv	512	$3 \times 3/2$	1		26×26
Conv	256	$1 \times 1/1$	Conv		26×26
Conv	512	$3 \times 3/1$	Conv	$\times 8$	26×26
Residual			Residual		26×26
Conv	1024	$3 \times 3/2$	1		13×13
Conv	512	$1 \times 1/1$	Conv		13×13
Conv	1024	$3 \times 3/1$	Conv	$\times 4$	13×13
Residual			Residual		13×13

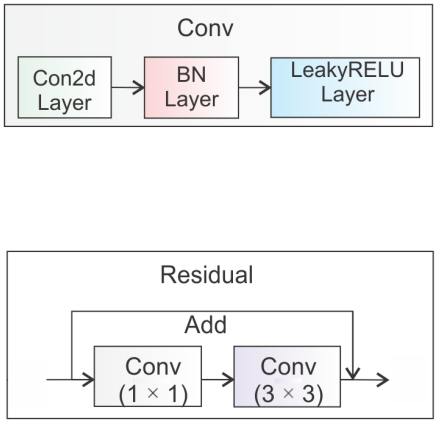

Figure 8: YOLOv3 Darknet-53 backbone. The architecture of YOLOv3 is composed of 53 convolutional layers, each with batch normalization and Leaky ReLU activation. Also, residual connections connect the input of the 1×1 convolutions across the whole network with the output of the 3×3 convolutions. The architecture shown here consists of only the backbone; it does not include the detection head composed of multi-scale predictions.

6.2 YOLOv3 Multi-Scale Predictions

Besides a larger architecture, an essential feature of YOLOv3 is the multi-scale predictions, i.e., predictions at multiple grid sizes. This helped to obtain finer detailed boxes and significantly improved the prediction of small objects, which was one of the main weaknesses of the previous versions of YOLO.

The multi-scale detection architecture shown in Figure 9 works as follows: the first output marked as **y1** is equivalent to the YOLOv2 output, where a 13×13 grid defines the output. The second output **y2** is composed by concatenating the output after the $(Res \times 4)$ of Darknet-53 with the output after (the $Res \times 8$). The feature maps have different sizes, i.e., 13×13 and 26×26, so there is an upsampling operation before the concatenation. Finally, using an upsampling operation, the third output **y3** concatenates the 26×26 feature maps with the 52×52 feature maps.

For the COCO dataset with 80 categories, each scale provides an output tensor with a shape of $N \times N \times [3 \times (4+1+80)]$ where $N \times N$ is the size of the feature map (or grid cell), the 3 indicates the boxes per cell and the $4+1$ include the four coordinates and the objectness score.

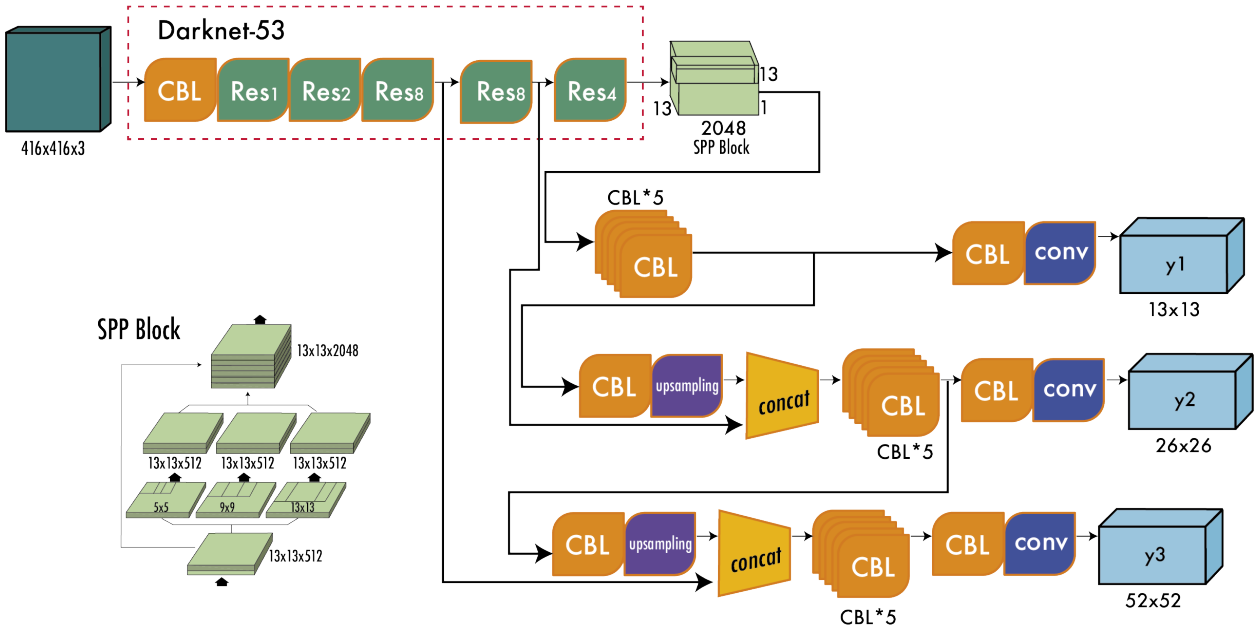

Figure 9: YOLOv3 Multi-scale detection architecture. The output of the Darknet-53 backbone is branched to three different outputs marked as **y1**, **y2**, and **y3**, each of increased resolution. The final predicted boxes are filtered using Non-maximum suppression. The CBL (Convolution-BatchNorm-Leaky ReLU) blocks comprise one convolution layer with batch normalization and leaky ReLU. The Res blocks comprise one CBL followed by two CBL structures with a residual connection, as shown in Figure 8.

6.3 YOLOv3 Results

When YOLOv3 was released, the benchmark for object detection had changed from PASCAL VOC to Microsoft COCO [37]. Therefore, from here on, all the YOLOs are evaluated in the MS COCO dataset. YOLOv3-spp achieved an average precision AP of 36.2% and AP_{50} of 60.6% at 20 FPS, achieving state-of-the-art at the time and $2\times$ faster.

7 Backbone, Neck, and Head

At this time, the architecture of object detectors started to be described in three parts: the backbone, the neck, and the head. Figure 10 shows a high-level backbone, neck, and head diagram.

The backbone is responsible for extracting useful features from the input image. It is typically a convolutional neural network (CNN) trained on a large-scale image classification task, such as ImageNet. The backbone captures hierarchical features at different scales, with lower-level features (e.g., edges and textures) extracted in the earlier layers and higher-level features (e.g., object parts and semantic information) extracted in the deeper layers.

The neck is an intermediate component that connects the backbone to the head. It aggregates and refines the features extracted by the backbone, often focusing on enhancing the spatial and semantic information across different scales. The neck may include additional convolutional layers, feature pyramid networks (FPN) [49], or other mechanisms to improve the representation of the features.

The head is the final component of an object detector; it is responsible for making predictions based on the features provided by the backbone and neck. It typically consists of one or more task-specific subnetworks that perform classification, localization, and, more recently, instance segmentation and pose estimation. The head processes the features the neck provides, generating predictions for each object candidate. In the end, a post-processing step, such as non-maximum suppression (NMS), filters out overlapping predictions and retains only the most confident detections.

In the rest of the YOLO models, we will describe the architectures using the backbone, neck, and head.

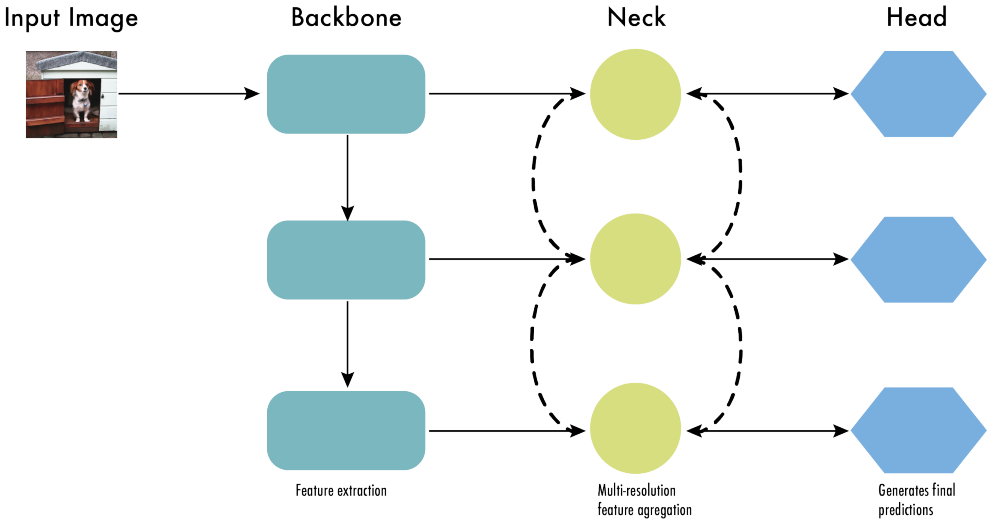

Figure 10: The architecture of modern object detectors can be described as the backbone, the neck, and the head. The backbone, usually a convolutional neural network (CNN), extracts vital features from the image at different scales. The neck refines these features, enhancing spatial and semantic information. Lastly, the head uses these refined features to make object detection predictions.

8 YOLOv4

Two years passed, and there was no new version of YOLO. It was until April 2020 that Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao released in ArXiv the paper for YOLOv4 [50]. At first, it felt odd that different authors presented a new "official" version of YOLO; however, YOLOv4 kept the same YOLO philosophy —real-time, open source, single shot, and darknet framework— and the improvements were so satisfactory that the community rapidly embrace this version as the official YOLOv4.

YOLOv4 tried to find the optimal balance by experimenting with many changes categorized as *bag-of-freebies* and *bag-of-specials*. Bag-of-freebies are methods that only change the training strategy and increase training cost but do not increase the inference time, the most common being data augmentation. On the other hand, bag-of-specials are methods that slightly increase the inference cost but significantly improve accuracy. Examples of these methods are those for enlarging the receptive field [48, 51, 52], combining features [53, 49, 54, 55], and post-processing [56, 40, 57, 58] among others.

We summarize the main changes of YOLOv4 in the following points:

- **An Enhanced Architecture with Bag-of-Specials (BoS) Integration**. The authors tried multiple architectures for the backbone, such as ResNeXt50 [59], EfficientNet-B3 [60], and Darknet-53. The best-performing architecture was a modification of Darknet-53 with cross-stage partial connections (CSPNet) [61], and Mish activation function [57] as the backbone (see Figure 11. For the neck, they used the modified version of spatial pyramid pooling (SPP) [48] from YOLOv3-spp and multi-scale predictions as in YOLOv3, but with a modified version of path aggregation network (PANet) [62] instead of FPN as well as a modified spatial attention module (SAM) [63]. Finally, for the detection head, they use anchors as in YOLOv3. Therefore, the model was called *CSPDarknet53-PANet-SPP*. The cross-stage partial connections (CSP) added to the Darknet-53 help reduce the computation of the model while keeping the same accuracy. The SPP block, as in YOLOv3-spp increases the receptive field without affecting the inference speed. The modified version of PANet concatenates the features instead of adding them as in the original PANet paper.
- **Integrating bag-of-freebies (BoF) for an Advanced Training Approach**. Apart from the regular augmentations such as random brightness, contrast, scaling, cropping, flipping, and rotation, the authors implemented mosaic augmentation that combines four images into a single one allowing the detection of objects outside their usual context and also reducing the need for a large mini-batch size for batch normalization. For regularization, they used DropBlock [64] that works as a replacement of Dropout [65] but for convolutional neural networks

as well as class label smoothing [66, 67]. For the detector, they added CIoU loss [68] and Cross mini-bath normalization (CmBN) for collecting statistics from the entire batch instead of from single mini-batches as in regular batch normalization [69].

- **Self-adversarial Training (SAT)**. To make the model more robust to perturbations, an adversarial attack is performed on the input image to create a deception that the ground truth object is not in the image but keeps the original label to detect the correct object.

- **Hyperparameter Optimization with Genetic Algorithms**. To find the optimal hyperparameters used for training, they use genetic algorithms on the first 10% of periods, and a cosine annealing scheduler [70] to alter the learning rate during training. It starts reducing the learning rate slowly, followed by a quick reduction halfway through the training process ending with a slight reduction.

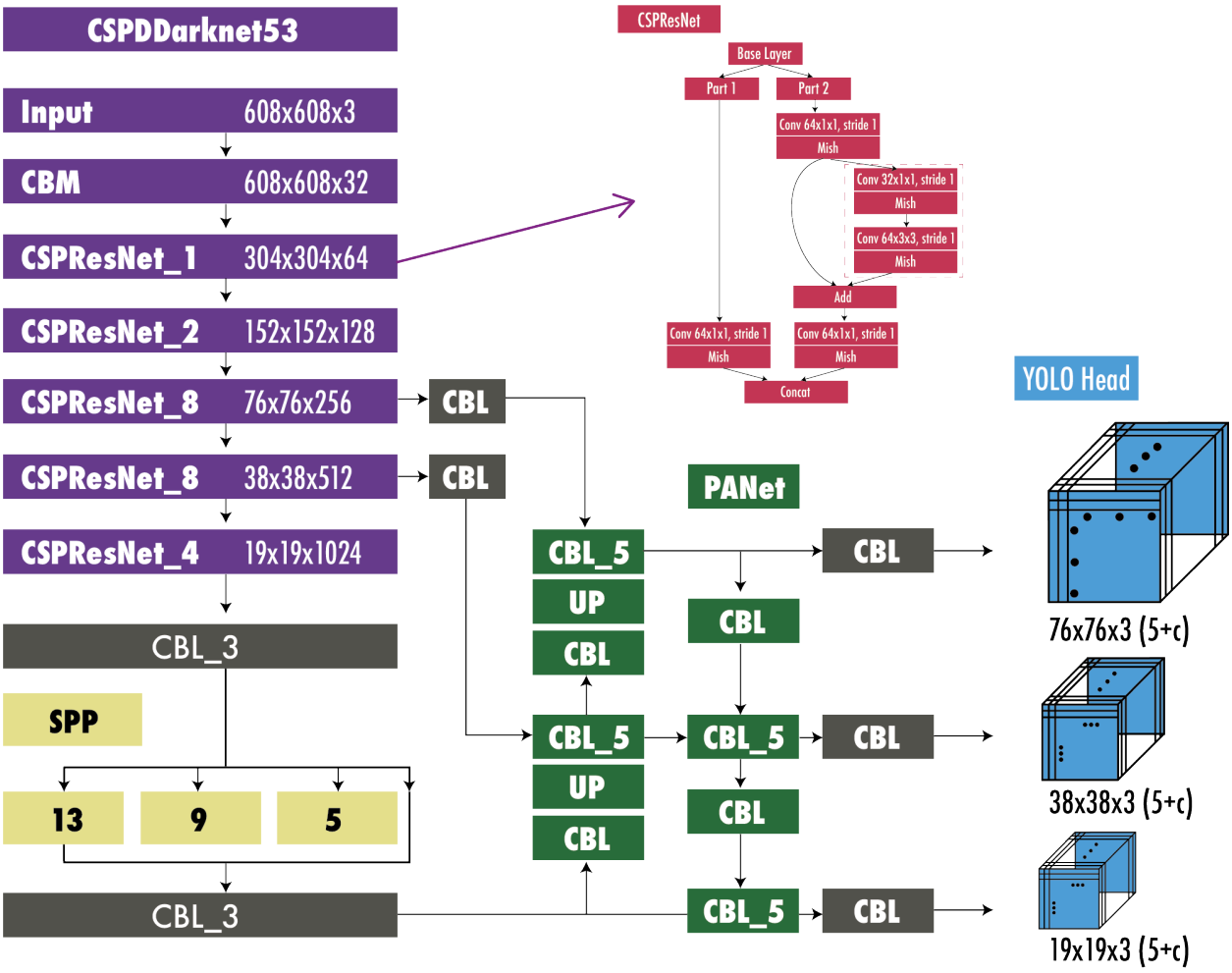

Figure 11: YOLOv4 Architecture for object detection. The modules in the diagram are **CMB**: Convolution + Batch Normalization + Mish activation, **CBL**: Convolution + Batch Normalization + Leaky ReLU, **UP**: upsampling, **SPP**: Spatial Pyramid Pooling, and **PANet**: Path Aggregation Network. Diagram inspired by [71].

Table 3 lists the final selection of BoFs and BoS for the backbone and the detector.

Evaluated on MS COCO dataset test-dev 2017, YOLOv4 achieved an AP of 43.5% and AP_{50} of 65.7% at more than 50 FPS on an NVIDIA V100.

Table 3: YOLOv4 final selection of bag-of-freebies (BoF) and bag-of-specials (BoS). BoF are methods that increase performance with no inference cost but longer training times. On the other hand, BoS are methods that slightly increase the inference cost but significantly improve accuracy.

Backbone	Detector
Bag-of-Freebies	**Bag-of-Freebies**
Data augmentation	Data augmentation
- Mosaic	- Mosaic
- CutMix	- Self-Adversarial Training
Regularization	CIoU loss
- DropBlock	Cross mini-Batch Normalization (CmBN)
Class label smoothing	Eliminate grid sensitivity
	Multiple anchors for a single ground truth
	Cosine annealing scheduler
	Optimal hyper-parameteres
	Random training shapes
Bag-of-Specials	**Bag-of-Specials**
Mish activation	Mish activation
Cross-stage partial connections	Spatial pyramid pooling block
Multi-input weighted residual connections	Spatial attention module (SAM)
	Path aggregation network (PAN)
	Distance-IoU Non-Maximum Suppression

9 YOLOv5

YOLOv5 [72] was released a couple of months after YOLOv4 in 2020 by Glen Jocher, founder and CEO of Ultralytics. It uses many improvements described in the YOLOv4 section but developed in Pytorch instead of Darknet. YOLOv5 incorporates an Ultralytics algorithm called AutoAnchor. This pre-training tool checks and adjusts anchor boxes if they are ill-fitted for the dataset and training settings, such as image size. It first applies a k-means function to dataset labels to generate initial conditions for a Genetic Evolution (GE) algorithm. The GE algorithm then evolves these anchors over 1000 generations by default, using CIoU loss [68] and Best Possible Recall as its fitness function. Figure 12 shows the detailed architecture of YOLOv5.

9.1 YOLOv5 Architecture

The backbone is a modified CSPDarknet53 that starts with a Stem, a strided convolution layer with a large window size to reduce memory and computational costs; followed by convolutional layers that extract relevant features from the input image. The SPPF (spatial pyramid pooling fast) layer and the following convolution layers process the features at various scales, while the upsample layers increase the resolution of the feature maps. The SPPF layer aims to speed up the computation of the network by pooling features of different scales into a fixed-size feature map. Each convolution is followed by batch normalization (BN) and SiLU activation [73]. The neck uses SPPF and a modified CSP-PAN, while the head resembles YOLOv3.

YOLOv5 uses several augmentations such as Mosaic, copy paste [74], random affine, MixUp [75], HSV augmentation, random horizontal flip, as well as other augmentations from the albumentations package [76]. It also improves the grid sensitivity to make it more stable to runaway gradients.

YOLOv5 provides five scaled versions: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large), where the width and depth of the convolution modules vary to suit specific applications and hardware requirements. For instance, YOLOv5n and YOLOv5s are lightweight models targeted for low-resource devices, while YOLOv5x is optimized for high performance, albeit at the expense of speed.

The YOLOv5 released version at the time of this writing is v7.0, including YOLOv5 versions capable of classification and instance segmentation.

YOLOv5 is open source and actively maintained by Ultralytics, with more than 250 contributors and new improvements frequently. YOLOv5 is easy to use, train and deploy. Ultralytics provide a mobile version for iOS and Android and many integrations for labeling, training, and deployment.

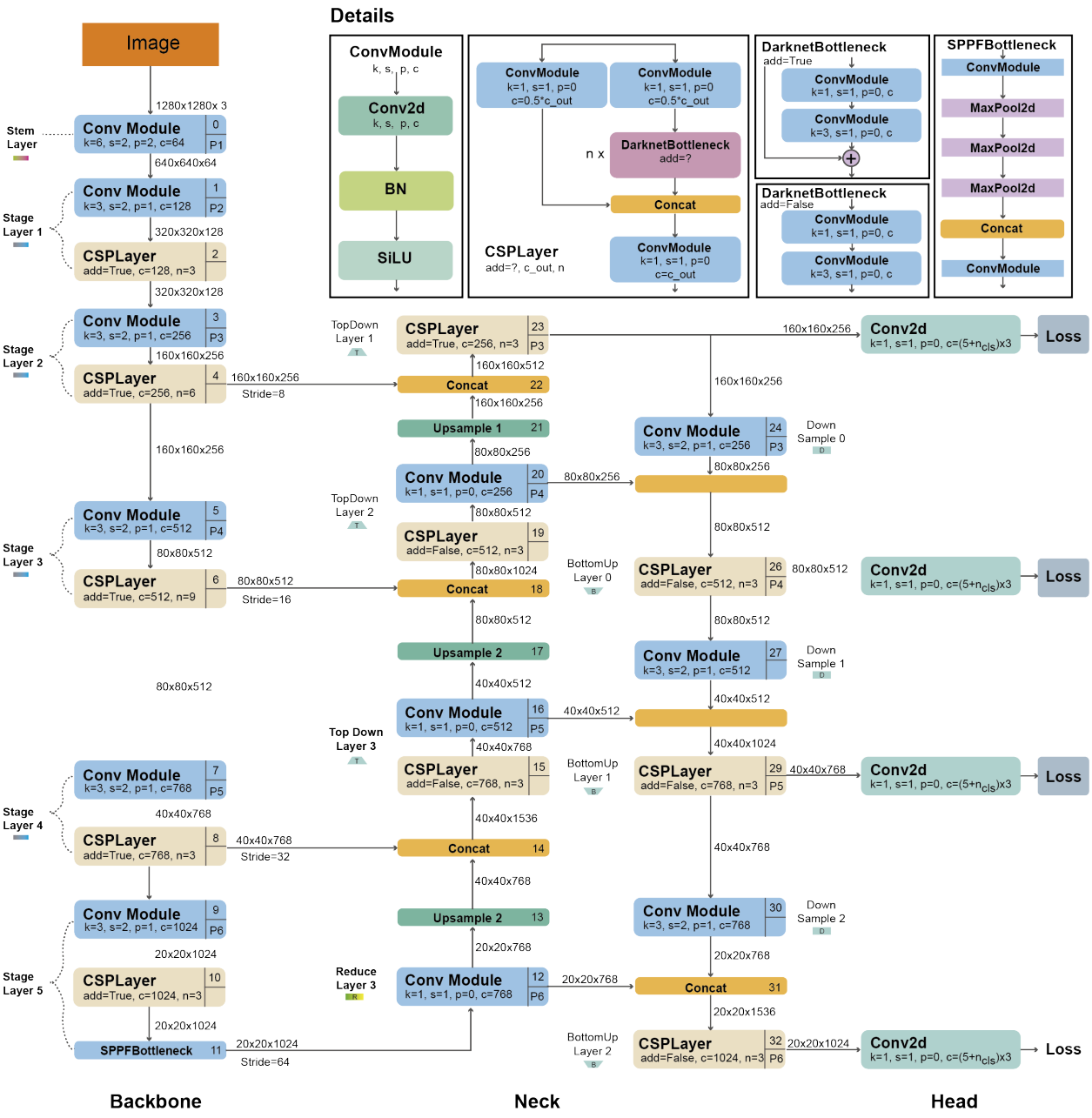

Figure 12: YOLOv5 Architecture. The architecture uses a modified CSPDarknet53 backbone with a Stem, followed by convolutional layers that extract image features. A spatial pyramid pooling fast (SPPF) layer accelerates computation by pooling features into a fixed-size map. Each convolution has batch normalization and SiLU activation. The network's neck uses SPPF and a modified CSP-PAN, while the head resembles YOLOv3. Diagram based in [77] and [78].

Evaluated on MS COCO dataset test-dev 2017, YOLOv5x achieved an AP of 50.7% with an image size of 640 pixels. Using a batch size of 32, it can achieve a speed of 200 FPS on an NVIDIA V100. Using a larger input size of 1536 pixels and test-time augmentation (TTA), YOLOv5 achieves an AP of 55.8%.

10 Scaled-YOLOv4

One year after YOLOv4, the same authors presented Scaled-YOLOv4 [79] in CVPR 2021. Differently from YOLOv4, Scaled YOLOv4 was developed in Pytorch instead of Darknet. The main novelty was the introduction of scaling-up and scaling-down techniques. Scaling up means producing a model that increases accuracy at the expense of a lower speed; on the other hand, scaling down entails producing a model that increases speed sacrificing accuracy. In addition, scaled-down models need less computing power and can run on embedded systems.

The scaled-down architecture was called YOLOv4-tiny; it was designed for low-end GPUs and can run at 46 FPS on a Jetson TX2 or 440 FPS on RTX2080Ti, achieving 22% AP on MS COCO.

The scaled-up model architecture was called YOLOv4-large, which included three different sizes P5, P6, and P7. This architecture was designed for cloud GPU and achieved state-of-the-art performance, surpassing all previous models [80, 81, 82] with 56% AP on MS COCO.

11 YOLOR

YOLOR [83] was published in ArXiv in May 2021 by the same research team of YOLOv4. It stands for *You Only Learn One Representation*. In this paper, the authors followed a different approach; they developed a multi-task learning approach that aims to create a single model for various tasks (e.g., classification, detection, pose estimation) by learning a general representation and using sub-networks to create task-specific representations. With the insight that the traditional joint learning method often leads to suboptimal feature generation, YOLOR aims to overcome this by encoding the implicit knowledge of neural networks to be applied to multiple tasks, similar to how humans use past experiences to approach new problems. The results showed that introducing implicit knowledge into the neural network benefits all the tasks.

Evaluated on MS COCO dataset test-dev 2017, YOLOR achieved a AP of 55.4% and AP_{50} of 73.3% at 30 FPS on an NVIDIA V100.

12 YOLOX

YOLOX [84] was published in ArXiv in July 2021 by Megvii Technology. Developed in Pytorch and using YOLOV3 from Ultralytics as starting point, it has five principal changes: an anchor-free architecture, multiple positives, a decoupled head, advanced label assignment, and strong augmentations. It achieved state-of-the-art results in 2021 with an optimal balance between speed and accuracy with 50.1% AP at 68.9% FPS on Tesla V100. In the following, we describe the five main changes of YOLOX with respect to YOLOv3:

1. **Anchor-free**. Since YOLOv2, all subsequent YOLO versions were anchor-based detectors. YOLOX, inspired by anchor-free state-of-the-art object detectors such as CornerNet [85], CenterNet [86], and FCOS [87], returned to an anchor-free architecture simplifying the training and decoding process. The anchor-free increased the AP by 0.9 points concerning the YOLOv3 baseline.

2. **Multi positives**. To compensate for the large imbalances the lack of anchors produced, the authors use center sampling [87] where they assigned the center 3×3 area as positives. This approach increased AP by 2.1 points.

3. **Decoupled head**. In [88, 89], it was shown that there could be a misalignment between the classification confidence and localization accuracy. Due to this, YOLOX separates these two into two heads (as shown in Fig. 13), one for classification tasks and the other for regression tasks improving the AP by 1.1 points and speeding up the model convergence.

4. **Advanced label assignment**. In [90], it was shown that the ground truth label assignment could have ambiguities when the boxes of multiple objects overlap and formulate the assigning procedure as an Optimal Transport (OT) problem. YOLOX, inspired by this work, proposed a simplified version called simOTA. This change increased AP by 2.3 points.

5. **Strong augmentations**. YOLOX uses MixUP [75] and Mosaic augmentations. The authors found that ImageNet pretraining was no longer beneficial after using these augmentations. The strong augmentations increased AP by 2.4 points.

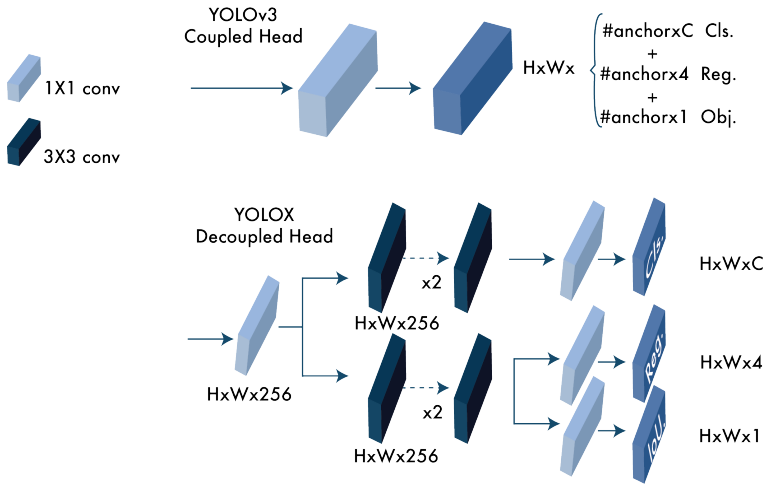

Figure 13: Difference between YOLOv3 head and YOLOX decoupled head. For each level of the FPN, they used a 1×1 convolution layer to reduce the feature channel to 256 and then added two parallel branches with two 3×3 convolution layers each for the class confidence (classification) and localization (regression) tasks. The IoU branch is added to the regression head.

13 YOLOv6

YOLOv6 [91] was published in ArXiv in September 2022 by Meituan Vision AI Department. The network design consists of an efficient backbone with RepVGG or CSPStackRep blocks, a PAN topology neck, and an efficient decoupled head with a hybrid-channel strategy. In addition, the paper introduces enhanced quantization techniques using post-training quantization and channel-wise distillation, resulting in faster and more accurate detectors. Overall, YOLOv6 outperforms previous state-of-the-art models on both accuracy and speed metrics, such as YOLOv5, YOLOX, and PP-YOLOE.

The main novelties of this model are summarized below:

1. **A new backbone based on RepVGG** [92] called EfficientRep that uses higher parallelism than previous YOLO backbones. For the neck, they use PAN [62] enhanced with RepBlocks [92] or CSPStackRep[61] Blocks for the larger models. And inspired by YOLOX, they developed an efficient decoupled head.
2. **Label assignment** using the Task alignment learning approach introduced in TOOD [93].
3. **New classification and regression losses**. They used a classification VariFocal loss [94] and a SIoU [95]/GIoU [96] regression loss.
4. **A self-distillation** strategy for the regression and classification tasks.
5. **A quantization scheme** for detection using RepOptimizer[97] and channel-wise distillation [98] that helped to achieve a faster detector.

Evaluated on MS COCO dataset test-dev 2017, YOLOv6-L achieved an AP of 52.5% and AP_{50} of 70% at around 50 FPS on an NVIDIA Tesla T4.

14 YOLOv7

YOLOv7 [99] was published in ArXiv in July 2022 by the same authors of YOLOv4 and YOLOR. At the time, it surpassed all known object detectors in speed and accuracy in the range of 5 FPS to 160 FPS. Like YOLOv4, it was trained using only the MS COCO dataset without pre-trained backbones. YOLOv7 proposed a couple of architecture changes and a series of bag-of-freebies, which increased the accuracy without affecting the inference speed, only the training time.

The architecture changes of YOLOv7 are:

- **Extended efficient layer aggregation network (E-ELAN)**. ELAN [100] is a strategy that allows a deep model to learn and converge more efficiently by controlling the shortest longest gradient path. YOLOv7

proposed E-ELAN that works for models with unlimited stacked computational blocks. E-ELAN combines the features of different groups by shuffling and merging cardinality to enhance the network's learning without destroying the original gradient path.

- **Model scaling for concatenation-based models**. Scaling generates models of different sizes by adjusting some model attributes. The architecture of YOLOv7 is a concatenation-based architecture in which standard scaling techniques, such as depth scaling, cause a ratio change between the input channel and the output channel of a transition layer which, in turn, leads to a decrease in the hardware usage of the model. YOLOv7 proposed a new strategy for scaling concatenation-based models in which the depth and width of the block are scaled with the same factor to maintain the optimal structure of the model.

The bag-of-freebies used in YOLOv7 include:

- **Planned re-parameterized convolution**. Like YOLOv6, the architecture of YOLOv7 is also inspired by re-parameterized convolutions (RepConv) [92]. However, they found that the identity connection in RepConv destroys the residual in ResNet [53] and the concatenation in DenseNet [101]. For this reason, they removed the identity connection and called it RepConvN.
- **Coarse label assignment for auxiliary head and fine label assignment for the lead head**. The lead head is responsible for the final output, while the auxiliary head assists with the training.
- **Batch normalization in conv-bn-activation**. This integrates the mean and variance of batch normalization into the bias and weight of the convolutional layer at the inference stage.
- **Implicit knowledge** inspired in YOLOR [83].
- **Exponential moving average** as the final inference model.

14.1 Comparison with YOLOv4 and YOLOR

In this section, we highlight the enhancements of YOLOv7 compared to previous YOLO models developed by the same authors.

Compared to YOLOv4, YOLOv7 achieved a 75% reduction in parameters and a 36% reduction in computation while simultaneously improving the average precision (AP) by 1.5%.

In contrast to YOLOv4-tiny, YOLOv7-tiny managed to reduce parameters and computation by 39% and 49%, respectively, while maintaining the same AP.

Lastly, compared to YOLOR, YOLOv7 reduced the number of parameters and computation by 43% and 15%, respectively, along with a slight 0.4% increase in AP.

Evaluated on MS COCO dataset test-dev 2017, YOLOv7-E6 achieved an AP of 55.9% and AP_{50} of 73.5% with an input size of 1280 pixels with a speed of 50 FPS on an NVIDIA V100.

15 DAMO-YOLO

DAMO-YOLO [102] was published in ArXiv in November 2022 by Alibaba Group. Inspired by the current technologies, DAMO-YOLO included the following:

1. **A Neural architecture search (NAS)**. They used a method called MAE-NAS [103] developed by Alibaba to find an efficient architecture automatically.
2. **A large neck**. Inspired by GiraffeDet [104], CSPNet [61], and ELAN [100], the authors designed a neck that can work in real-time called Efficient-RepGFPN.
3. **A small head**. The authors found that a large neck and a small neck yield better performance, and they only left one linear layer for classification and one for regression. They called this approach ZeroHead.
4. **AlignedOTA label assignment**. Dynamic label assignment methods, such as OTA[90] and TOOD[93], have gained popularity due to their significant improvements over static methods. However, the misalignment between classification and regression remains a problem, partly because of the imbalance between classification and regression losses. To address this issue, their AlignOTA method introduces focal loss [81] into the classification cost and uses the IoU of prediction and ground truth box as the soft label, enabling the selection of aligned samples for each target and solving the problem from a global perspective.

5. **Knowledge distillation**. Their proposed strategy consists of two stages: the teacher guiding the student in the first stage and the student fine-tuning independently in the second stage. Additionally, they incorporate two enhancements in the distillation approach: the Align Module, which adapts student features to the same resolution as the teacher's, and Channel-wise Dynamic Temperature, which normalizes teacher and student features to reduce the impact of real value differences.

The authors generated scaled models named DAMO-YOLO-Tiny/Small/Medium, with the best model achieving an AP of 50.0 % at 233 FPS on an NVIDIA V100.

16 YOLOv8

YOLOv8 [105] was released in January 2023 by Ultralytics, the company that developed YOLOv5. YOLOv8 provided five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) and YOLOv8x (extra large). YOLOv8 supports multiple vision tasks such as object detection, segmentation, pose estimation, tracking, and classification.

16.1 YOLOv8 Architecture

Figure 14 shows the detailed architecture of YOLOv8. YOLOv8 uses a similar backbone as YOLOv5 with some changes on the CSPLayer, now called the C2f module. The C2f module (cross-stage partial bottleneck with two convolutions) combines high-level features with contextual information to improve detection accuracy.

YOLOv8 uses an anchor-free model with a decoupled head to independently process objectness, classification, and regression tasks. This design allows each branch to focus on its task and improves the model's overall accuracy. In the output layer of YOLOv8, they used the sigmoid function as the activation function for the objectness score, representing the probability that the bounding box contains an object. It uses the softmax function for the class probabilities, representing the objects' probabilities belonging to each possible class.

YOLOv8 uses CIoU [68] and DFL [106] loss functions for bounding box loss and binary cross-entropy for classification loss. These losses have improved object detection performance, particularly when dealing with smaller objects.

YOLOv8 also provides a semantic segmentation model called YOLOv8-Seg model. The backbone is a CSPDarknet53 feature extractor, followed by a C2f module instead of the traditional YOLO neck architecture. The C2f module is followed by two segmentation heads, which learn to predict the semantic segmentation masks for the input image. The model has similar detection heads to YOLOv8, consisting of five detection modules and a prediction layer. The YOLOv8-Seg model has achieved state-of-the-art results on various object detection and semantic segmentation benchmarks while maintaining high speed and efficiency.

YOLOv8 can be run from the command line interface (CLI), or it can also be installed as a PIP package. In addition, it comes with multiple integrations for labeling, training, and deploying.

Evaluated on MS COCO dataset test-dev 2017, YOLOv8x achieved an AP of 53.9% with an image size of 640 pixels (compared to 50.7% of YOLOv5 on the same input size) with a speed of 280 FPS on an NVIDIA A100 and TensorRT.

17 PP-YOLO, PP-YOLOv2, and PP-YOLOE

PP-YOLO models have been growing parallel to the YOLO models we described. However, we decided to group them in a single section because they began with YOLOv3 and had been gradually improving upon the previous PP-YOLO version. Nevertheless, these models have been influential in the evolution of YOLO. PP-YOLO [82] similar to YOLOv4 and YOLOv5 was based on YOLOv3. It was published in ArXiv in July 2020 by researchers from Baidu Inc. The authors used the PaddlePaddle [107] deep learning platform, hence its *PP* name. Following the trend we have seen starting with YOLOv4, PP-YOLO added ten existing tricks to improve the detector's accuracy, keeping the speed unchanged. According to the authors, this paper was not intended to introduce a novel object detector but to show how to build a better detector step by step. Most of the tricks PP-YOLO uses are different from the ones used in YOLOv4, and the ones that overlap use a different implementation. The changes of PP-YOLO concerning YOLOv3 are:

1. **A ResNet50-vd backbone** replacing the DarkNet-53 backbone with an architecture augmented with deformable convolutions [108] in the last stage and a distilled pre-trained model, which has a higher classification accuracy on ImageNet. This architecture was called ResNet5-vd-dcn.
2. **A larger batch size** to improve training stability, they went from 64 to 192, along with an updated training schedule and learning rate.

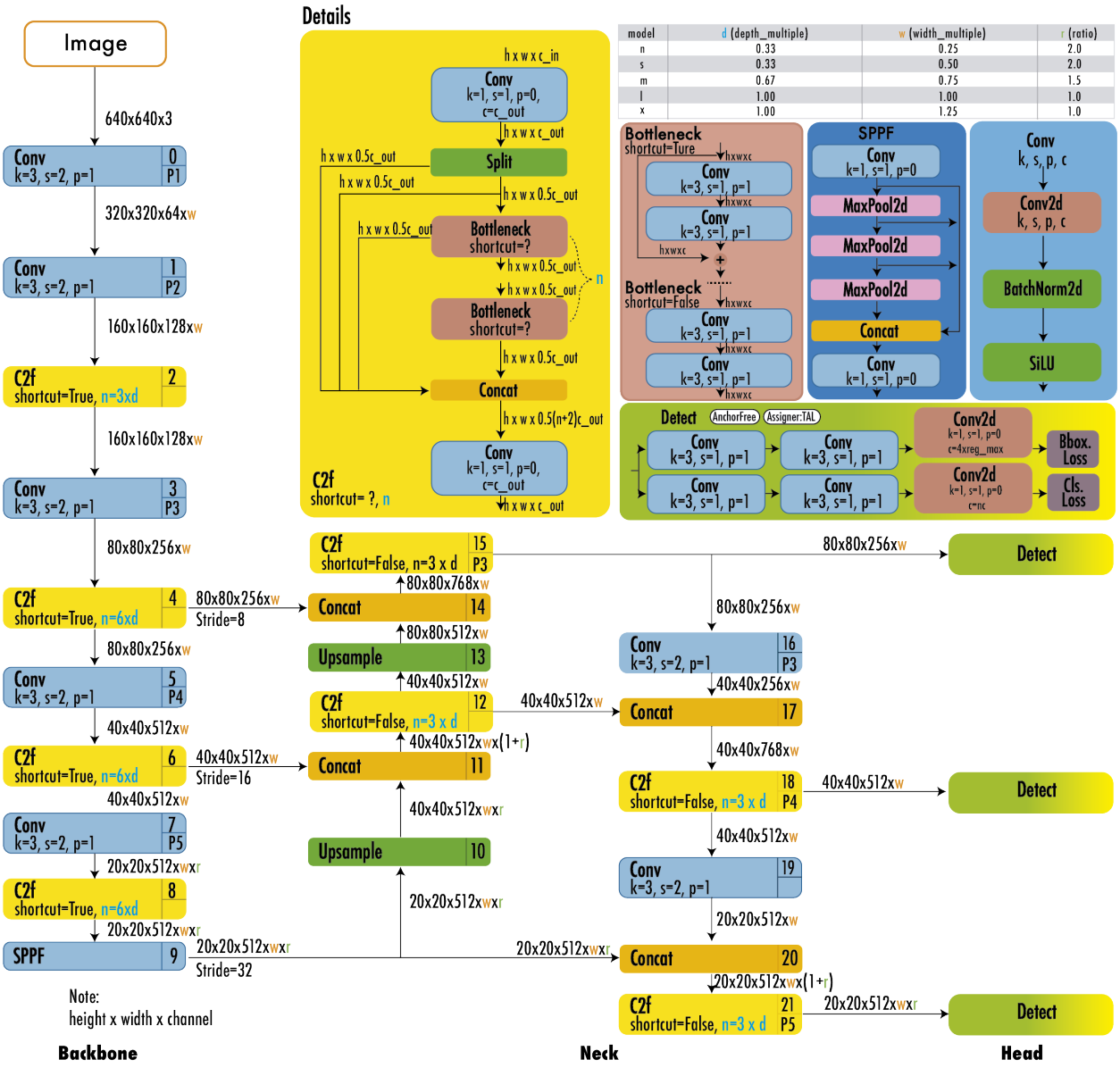

Figure 14: YOLOv8 Architecture. The architecture uses a modified CSPDarknet53 backbone. The C2f module replaces the CSPLayer used in YOLOv5. A spatial pyramid pooling fast (SPPF) layer accelerates computation by pooling features into a fixed-size map. Each convolution has batch normalization and SiLU activation. The head is decoupled to process objectness, classification, and regression tasks independently.

3. **Maintained moving averages** for the trained parameters and use them instead of the final trained values.
4. **DropBlock** is applied only to the FPN.
5. **An IoU loss** is added in another branch along with the L1-loss for bounding box regression.
6. **An IoU prediction branch** is added to measure localization accuracy along with an IoU aware loss. During inference, YOLOv3 multiplies the classification probability and objectiveness score to compute the final detection, PP-YOLO also multiplies the predicted IoU to consider the localization accuracy.
7. **Grid Sensitive approach** similar to YOLOv4 is used to improve the bounding box center prediction at the grid boundary.
8. **Matrix NMS** [109] is used, which can be run in parallel making it faster than traditional NMS.

9. **CoordConv** [110] is used for the 1×1 convolution of the FPN, and on the first convolution layer in the detection head. CoordConv allows the network to learn translational invariance improving the detection localization.
10. **Spatial Pyramid Pooling** is used only on the top feature map to increase the receptive field of the backbone.

17.1 PP-YOLO augmentations and preprocessing

PP-YOLO used the following augmentations and preprocessing:

1. Mixup Training [75] with a weight sampled from $Beta(\alpha, \beta)$ distribution where $\alpha = 1.5$ and $\beta = 1.5$.
2. Random Color Distortion.
3. Random Expand.
4. Random Crop and Random Flip with a probability of 0.5.
5. RGB channel z-score normalization with a mean of $[0.485, 0.456, 0.406]$ and a standard deviation of $[0.229, 0.224, 0.225]$.
6. Multiple image sizes evenly drawn from [320, 352, 384, 416, 448, 480, 512, 544, 576, 608].

Evaluated on MS COCO dataset test-dev 2017, PP-YOLO achieved an AP of 45.9% and AP_{50} of 65.2% at 73 FPS on an NVIDIA V100.

17.2 PP-YOLOv2

PP-YOLOv2 [111] was published in ArXiv on April 2021 and added four refinements to PP-YOLO that increased performance from 45.9% AP to 49.5% AP at 69 FPS on NVIDIA V100. The changes of PP-YOLOv2 concerning PP-YOLO are the following:

1. **Backbone changed from ResNet50 to ResNet101**.
2. **Path aggregation network (PAN)** instead of FPN similar to YOLOv4.
3. **Mish Activation Function**. Unlike YOLOv4 and YOLOv5, they only applied the mish activation function in the detection neck to keep the backbone unchanged with ReLU.
4. **Larger input sizes** help to increase performance on small objects. They expanded the largest input size from 608 to 768 and reduced the batch size from 24 to 12 images per GPU. The input sizes are evenly drawn from [320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768].
5. **A modified IoU aware branch**. They modified the calculation of the IoU aware loss calculation using a soft label format instead of a soft weight format.

17.3 PP-YOLOE

PP-YOLOE [112] was published in ArXiv in March 2022. It added improvements upon PP-YOLOv2 achieving a performance of 51.4% AP at 78.1 FPS on NVIDIA V100. The main changes of PP-YOLOE concerning PP-YOLOv2 are:

1. **Anchor-free**. Following the trends of the time driven by the works of [87, 86, 85, 84], PP-YOLOE uses an anchor-free architecture.
2. **New backbone and neck**. Inspired by TreeNet [113], the authors modified the architecture of the backbone and neck with RepResBlocks combining residual and dense connections.
3. **Task Alignment Learning (TAL)**. YOLOX was the first to bring up the problem of task misalignment, where the classification confidence and the location accuracy do not agree in all cases. To reduce this problem, PP-YOLOE implemented TAL as proposed in TOOD [93], which includes a dynamic label assignment combined with a task-alignment loss.
4. **Efficient Task-aligned Head (ET-head)**. Different from YOLOX where the classification and locations heads were decoupled, PP-YOLOE instead used a single head based on TOOD to improve speed and accuracy.

5. **Varifocal (VFL) and Distribution focal loss (DFL)**. VFL [94] weights loss of positive samples using target score, giving higher weight to those with high IoU. This prioritizes high-quality samples during training. Similarly, both use IoU-aware classification score (IACS) as the target, allowing for joint learning of classification and localization quality, leading to consistency between training and inference. On the other hand, DFL [106] extends Focal Loss from discrete to continuous labels, enabling successful optimization of improved representations that combine quality estimation and class prediction. This allows for an accurate depiction of flexible distribution in real data, eliminating the risk of inconsistency.

Like previous YOLO versions, the authors generated multiple scaled models by varying the width and depth of the backbone and neck. The models are called PP-YOLOE-s (small), PP-YOLOE-m (medium), PP-YOLOE-l (large), and PP-YOLOE-x (extra large).

18 YOLO-NAS

YOLO-NAS [114] was released in May 2023 by Deci, a company that develops production-grade models and tools to build, optimize, and deploy deep learning models. YOLO-NAS is designed to detect small objects, improve localization accuracy, and enhance the performance-per-compute ratio, making it suitable for real-time edge-device applications. In addition, its open-source architecture is available for research use.

The novelty of YOLO-NAS includes the following:

- Quantization aware modules [115] called QSP and QCI that combine re-parameterization for 8-bit quantization to minimize the accuracy loss during post-training quantization.
- Automatic architecture design using AutoNAC, Deci's proprietary NAS technology.
- Hybrid quantization method to selectively quantize certain parts of a model to balance latency and accuracy instead of standard quantization, where all the layers are affected.
- A pre-training regimen with automatically labeled data, self-distillation, and large datasets.

The AutoNAC system, which was instrumental in creating YOLO-NAS, is versatile and can accommodate any task, the specifics of the data, the environment for making inferences, and the setting of performance goals. It assists users in identifying the most suitable structure that offers the perfect blend of precision and inference speed for their particular use. This technology considers the data and hardware and other elements involved in the inference process, such as compilers and quantization. In addition, RepVGG blocks were incorporated into the model architecture during the NAS process for compatibility with Post-Training Quantization (PTQ). They generated three architectures by varying the depth and positions of the QSP and QCI blocks: YOLO-NASS, YOLO-NASM, and YOLO-NASL (S,M,L for small, medium, and large, respectively). Figure 15 shows the model architecture for YOLO-NASL.

The model is pre-trained on Objects365 [116], which contains two million images and 365 categories, then the COCO dataset was used to generate pseudo-labels. Finally, the models are trained with the original 118k train images of the COCO dataset.

At this writing, three YOLO-NAS models have been released in FP32, FP16, and INT8 precisions achieving an AP of 52.2% on MS COCO with 16-bit precision.

19 Discussion

This paper examined 16 YOLO versions, ranging from the original YOLO model to the most recent YOLO-NAS. Table 4 provides an overview of the YOLO versions discussed. From this table, we can identify several key patterns:

- **Anchors**: The original YOLO model was relatively simple and did not employ anchors, while the state-of-the-art relied on two-stage detectors with anchors. YOLOv2 incorporated anchors, leading to improvements in bounding box prediction accuracy. This trend persisted for five years until YOLOX introduced an anchor-less approach that achieved state-of-the-art results. Since then, subsequent YOLO versions have abandoned the use of anchors.
- **Framework**: Initially, YOLO was developed using the Darknet framework, with subsequent versions following suit. However, when Ultralytics ported YOLOv3 to PyTorch, the remaining YOLO versions were developed using PyTorch, leading to a surge in enhancements. Another deep learning language utilized is PaddlePaddle, an open-source framework initially developed by Baidu.

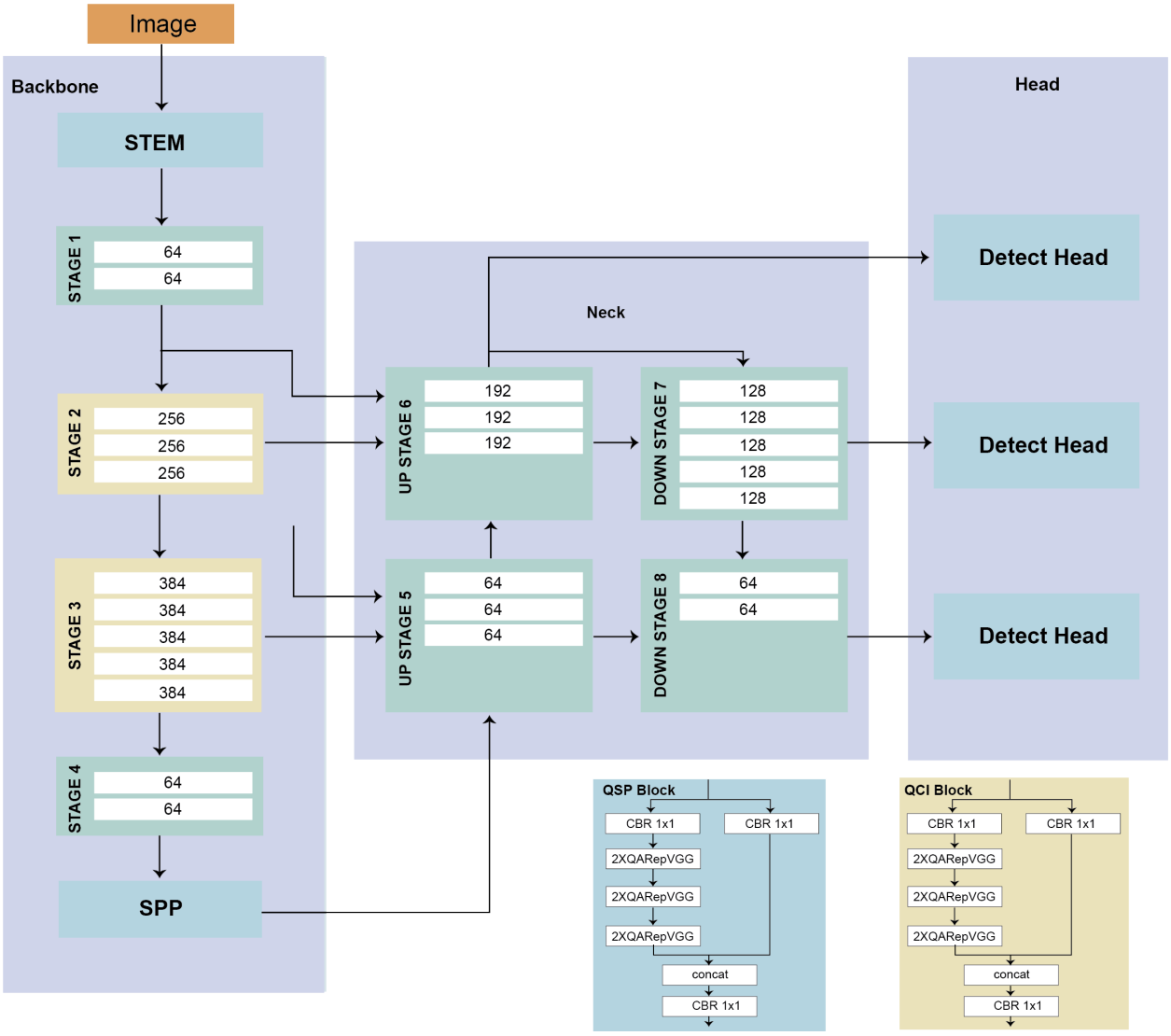

Figure 15: YOLO-NAS Architecture. The architecture is found automatically via a Neural Architecture Search (NAS) system called AutoNAC to balance latency vs. throughput. They generated three architectures called YOLO-NASS (small), YOLO-NASM (medium), and YOLO-NASL (large), varying the depth and positions of the QSP and QCI blocks. The figure shows the YOLO-NASL architecture.

- **Backbone**: The backbone architectures of YOLO models have undergone significant changes over time. Starting with the Darknet architecture, which comprised simple convolutional and max pooling layers, later models incorporated cross-stage partial connections (CSP) in YOLOv4, reparameterization in YOLOv6 and YOLOv7, and neural architecture search in DAMO-YOLO and YOLO-NAS.
- **Performance**: While the performance of YOLO models has improved over time, it is worth noting that they often prioritize balancing speed and accuracy rather than solely focusing on accuracy. This tradeoff is essential to the YOLO framework, allowing for real-time object detection across various applications.

19.1 Tradeoff between speed and accuracy

The YOLO family of object detection models has consistently focused on balancing speed and accuracy, aiming to deliver real-time performance without sacrificing the quality of detection results. As the YOLO framework has evolved through its various iterations, this tradeoff has been a recurring theme, with each version seeking to optimize these

Table 4: Summary of YOLO architectures. The metric reported for YOLO and YOLOv2 were on VOC2007, while the rest are reported on COCO2017. The NAS-YOLO model reported has 16-bit precision.

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*63.4*
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

competing objectives differently. In the original YOLO model, the primary focus was on achieving high-speed object detection. The model utilized a single convolutional neural network (CNN) to directly predict object locations and classes from the input image, enabling real-time processing. However, this emphasis on speed led to a compromise in accuracy, mainly when dealing with small objects or objects with overlapping bounding boxes.

Subsequent YOLO versions introduced refinements and enhancements to address these limitations while maintaining the framework's real-time capabilities. For instance, YOLOv2 (YOLO9000) introduced anchor boxes and passthrough layers to improve the localization of objects, resulting in higher accuracy. In addition, YOLOv3 enhanced the model's performance by employing a multi-scale feature extraction architecture, allowing for better object detection across various scales.

The tradeoff between speed and accuracy became more nuanced as the YOLO framework evolved. Models like YOLOv4 and YOLOv5 introduced innovations, such as new network backbones, improved data augmentation techniques, and optimized training strategies. These developments led to significant gains in accuracy without drastically affecting the models' real-time performance.

From YOLOv5, all official YOLO models have fine-tuned the tradeoff between speed and accuracy, offering different model scales to suit specific applications and hardware requirements. For instance, these versions often provide lightweight models optimized for edge devices, trading accuracy for reduced computational complexity and faster processing times.

20 The future of YOLO

As the YOLO framework continues to evolve, we anticipate that the following trends and possibilities will shape future developments:

Incorporation of Latest Techniques. Researchers and developers will continue to refine the YOLO architecture by leveraging state-of-the-art methods in deep learning, data augmentation, and training techniques. This ongoing innovation process will likely improve the model's performance, robustness, and efficiency.

Benchmark Evolution. The current benchmark for evaluating object detection models, COCO 2017, may eventually be replaced by a more advanced and challenging benchmark. This mirrors the transition from the VOC 2007 benchmark used in the first two YOLO versions, reflecting the need for more demanding benchmarks as models grow more sophisticated and accurate.

Proliferation of YOLO Models and Applications. As the YOLO framework progresses, we expect to witness an increase in the number of YOLO models released each year, along with a corresponding expansion of applications. As the framework becomes more versatile and powerful, it will likely be employed in more varied domains, from home appliances devices to autonomous cars.

Expansion into New Domains. YOLO models have the potential to expand beyond object detection and segmentation, exploring domains such as object tracking in videos and 3D keypoint estimation. In the future, we anticipate YOLO models to transition into multi-modal frameworks, incorporating both vision and language, video, and sound processing. As these models evolve, they may serve as the foundation for innovative solutions catering to a broader spectrum of computer vision and multimedia tasks.

Adaptability to Diverse Hardware. YOLO models will further span hardware platforms, from IoT devices to high-performance computing clusters. This adaptability will enable deploying YOLO models in various contexts, depending on the application's requirements and constraints. In addition, by tailoring the models to suit different hardware specifications, YOLO can be made accessible and effective for more users and industries.

21 Acknowledgments

We wish to thank the National Council for Science and Technology (CONACYT) for its support through the National Research System (SNI).

References

[1] W. Lan, J. Dang, Y. Wang, and S. Wang, "Pedestrian detection based on yolo network model," in *2018 IEEE international conference on mechatronics and automation (ICMA)*, pp. 1547–1551, IEEE, 2018.

[2] W.-Y. Hsu and W.-Y. Lin, "Adaptive fusion of multi-scale yolo for pedestrian detection," *IEEE Access*, vol. 9, pp. 110063–110073, 2021.

[3] A. Benjumea, I. Teeti, F. Cuzzolin, and A. Bradley, "Yolo-z: Improving small object detection in yolov5 for autonomous vehicles," *arXiv preprint arXiv:2112.11798*, 2021.

[4] N. M. A. A. Dazlee, S. A. Khalil, S. Abdul-Rahman, and S. Mutalib, "Object detection for autonomous vehicles with sensor-based technology using yolo," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 1, pp. 129–134, 2022.

[5] S. Liang, H. Wu, L. Zhen, Q. Hua, S. Garg, G. Kaddoum, M. M. Hassan, and K. Yu, "Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 25345–25360, 2022.

[6] Q. Li, X. Ding, X. Wang, L. Chen, J. Son, and J.-Y. Song, "Detection and identification of moving objects at busy traffic road based on yolo v4," *The Journal of the Institute of Internet, Broadcasting and Communication*, vol. 21, no. 1, pp. 141–148, 2021.

[7] S. Shinde, A. Kothari, and V. Gupta, "Yolo based human action recognition and localization," *Procedia computer science*, vol. 133, pp. 831–838, 2018.

[8] A. H. Ashraf, M. Imran, A. M. Qahtani, A. Alsufyani, O. Almutiry, A. Mahmood, M. Attique, and M. Habib, "Weapons detection for security and video surveillance using cnn and yolo-v5s," *CMC-Comput. Mater. Contin*, vol. 70, pp. 2761–2775, 2022.

[9] Y. Zheng and H. Zhang, "Video analysis in sports by lightweight object detection network under the background of sports industry development," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.

[10] H. Ma, T. Celik, and H. Li, "Fer-yolo: Detection and classification based on facial expressions," in *Image and Graphics: 11th International Conference, ICIG 2021, Haikou, China, August 6–8, 2021, Proceedings, Part I 11*, pp. 28–39, Springer, 2021.

[11] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, "Apple detection during different growth stages in orchards using the improved yolo-v3 model," *Computers and electronics in agriculture*, vol. 157, pp. 417–426, 2019.

[12] D. Wu, S. Lv, M. Jiang, and H. Song, "Using channel pruning-based yolo v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments," *Computers and Electronics in Agriculture*, vol. 178, p. 105742, 2020.

[13] M. Lippi, N. Bonucci, R. F. Carpio, M. Contarini, S. Speranza, and A. Gasparri, "A yolo-based pest detection system for precision agriculture," in *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 342–347, IEEE, 2021.

[14] W. Yang and Z. Jiachun, "Real-time face detection based on yolo," in *2018 1st IEEE international conference on knowledge innovation and invention (ICKII)*, pp. 221–224, IEEE, 2018.

[15] W. Chen, H. Huang, S. Peng, C. Zhou, and C. Zhang, "Yolo-face: a real-time face detector," *The Visual Computer*, vol. 37, pp. 805–813, 2021.

[16] M. A. Al-Masni, M. A. Al-Antari, J.-M. Park, G. Gi, T.-Y. Kim, P. Rivera, E. Valarezo, M.-T. Choi, S.-M. Han, and T.-S. Kim, "Simultaneous detection and classification of breast masses in digital mammograms via a deep learning yolo-based cad system," *Computer methods and programs in biomedicine*, vol. 157, pp. 85–94, 2018.

[17] Y. Nie, P. Sommella, M. O'Nils, C. Liguori, and J. Lundgren, "Automatic detection of melanoma with yolo deep convolutional neural networks," in *2019 E-Health and Bioengineering Conference (EHB)*, pp. 1–4, IEEE, 2019.

[18] H. M. Ünver and E. Ayan, "Skin lesion segmentation in dermoscopic images with combination of yolo and grabcut algorithm," *Diagnostics*, vol. 9, no. 3, p. 72, 2019.

[19] L. Tan, T. Huangfu, L. Wu, and W. Chen, "Comparison of retinanet, ssd, and yolo v3 for real-time pill identification," *BMC medical informatics and decision making*, vol. 21, pp. 1–11, 2021.

[20] L. Cheng, J. Li, P. Duan, and M. Wang, "A small attentional yolo model for landslide detection from satellite remote sensing images," *Landslides*, vol. 18, no. 8, pp. 2751–2765, 2021.

[21] M.-T. Pham, L. Courtrai, C. Friguet, S. Lefèvre, and A. Baussard, "Yolo-fine: One-stage detector of small objects under various backgrounds in remote sensing images," *Remote Sensing*, vol. 12, no. 15, p. 2501, 2020.

[22] Y. Qing, W. Liu, L. Feng, and W. Gao, "Improved yolo network for free-angle remote sensing target detection," *Remote Sensing*, vol. 13, no. 11, p. 2171, 2021.

[23] Z. Zakria, J. Deng, R. Kumar, M. S. Khokhar, J. Cai, and J. Kumar, "Multiscale and direction target detecting in remote sensing images via modified yolo-v4," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 1039–1048, 2022.

[24] P. Kumar, S. Narasimha Swamy, P. Kumar, G. Purohit, and K. S. Raju, "Real-time, yolo-based intelligent surveillance and monitoring system using jetson tx2," in *Data Analytics and Management: Proceedings of ICDAM*, pp. 461–471, Springer, 2021.

[25] K. Bhambani, T. Jain, and K. A. Sultanpure, "Real-time face mask and social distancing violation detection system using yolo," in *2020 IEEE Bangalore humanitarian technology conference (B-HTC)*, pp. 1–6, IEEE, 2020.

[26] J. Li, Z. Su, J. Geng, and Y. Yin, "Real-time detection of steel strip surface defects based on improved yolo detection network," *IFAC-PapersOnLine*, vol. 51, no. 21, pp. 76–81, 2018.

[27] E. N. Ukhwah, E. M. Yuniarno, and Y. K. Suprapto, "Asphalt pavement pothole detection using deep learning method based on yolo neural network," in *2019 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 35–40, IEEE, 2019.

[28] Y. Du, N. Pan, Z. Xu, F. Deng, Y. Shen, and H. Kang, "Pavement distress detection and classification based on yolo network," *International Journal of Pavement Engineering*, vol. 22, no. 13, pp. 1659–1672, 2021.

[29] R.-C. Chen *et al.*, "Automatic license plate recognition via sliding-window darknet-yolo deep learning," *Image and Vision Computing*, vol. 87, pp. 47–56, 2019.

[30] C. Dewi, R.-C. Chen, X. Jiang, and H. Yu, "Deep convolutional neural network for enhancing traffic sign recognition developed on yolo v4," *Multimedia Tools and Applications*, vol. 81, no. 26, pp. 37821–37845, 2022.

[31] A. M. Roy, J. Bhaduri, T. Kumar, and K. Raj, "Wildect-yolo: An efficient and robust computer vision-based accurate object localization model for automated endangered wildlife detection," *Ecological Informatics*, vol. 75, p. 101919, 2023.

[32] S. Kulik and A. Shtanko, "Experiments with neural net object detection system yolo on small training datasets for intelligent robotics," in *Advanced Technologies in Robotics and Intelligent Systems: Proceedings of ITR 2019*, pp. 157–162, Springer, 2020.

[33] D. H. Dos Reis, D. Welfer, M. A. De Souza Leite Cuadros, and D. F. T. Gamarra, "Mobile robot navigation using an object recognition software with rgbd images and the yolo algorithm," *Applied Artificial Intelligence*, vol. 33, no. 14, pp. 1290–1305, 2019.

[34] O. Sahin and S. Ozer, "Yolodrone: Improved yolo architecture for object detection in drone images," in *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 361–365, IEEE, 2021.

[35] C. Chen, Z. Zheng, T. Xu, S. Guo, S. Feng, W. Yao, and Y. Lan, "Yolo-based uav technology: A review of the research and its applications," *Drones*, vol. 7, no. 3, p. 190, 2023.

[36] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[37] 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 *European conference on computer vision*, pp. 740–755, Springer, 2014.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[39] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[40] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, Atlanta, Georgia, USA, 2013.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[42] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[44] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[45] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[46] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[47] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, *et al.*, "Openimages: A public dataset for large-scale multi-label and multi-class image classification," *Dataset available from https://github. com/openimages*, vol. 2, no. 3, p. 18, 2017.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[49] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[50] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[51] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[52] S. Liu, D. Huang, *et al.*, "Receptive field block net for accurate and fast object detection," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 385–400, 2018.

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

[54] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 447–456, 2015.

[55] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 9259–9266, 2019.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[57] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *arXiv preprint arXiv:1908.08681*, vol. 4, no. 2, pp. 10–48550, 2019.

[58] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms–improving object detection with one line of code," in *Proceedings of the IEEE international conference on computer vision*, pp. 5561–5569, 2017.

[59] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[60] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[61] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020.

[62] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[63] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[64] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," *Advances in neural information processing systems*, vol. 31, 2018.

[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[66] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[67] M. A. Islam, S. Naha, M. Rochan, N. Bruce, and Y. Wang, "Label refinement network for coarse-to-fine semantic segmentation," *arXiv preprint arXiv:1703.00551*, 2017.

[68] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 12993–13000, 2020.

[69] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[70] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[71] S. Wang, J. Zhao, N. Ta, X. Zhao, M. Xiao, and H. Wei, "A real-time deep learning forest fire monitoring algorithm based on an improved pruned+ kd model," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2319–2329, 2021.

[72] G. Jocher, "YOLOv5 by Ultralytics." `https://github.com/ultralytics/yolov5`, 2020. Accessed: February 30, 2023.

[73] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[74] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph, "Simple copy-paste is a strong data augmentation method for instance segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2918–2928, 2021.

[75] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[76] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020.

[77] M. Contributors, "YOLOv5 by MMYOLO." `https://github.com/open-mmlab/mmyolo/tree/main/configs/yolov5`, 2023. Accessed: May 13, 2023.

[78] Ultralytics, "Model Structure." `https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure`, 2023. Accessed: May 14, 2023.

[79] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pp. 13029–13038, 2021.

[80] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

[81] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[82] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, *et al.*, "Pp-yolo: An effective and efficient implementation of object detector," *arXiv preprint arXiv:2007.12099*, 2020.

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

[84] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[85] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 734–750, 2018.

[86] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6569–6578, 2019.

[87] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9627–9636, 2019.

[88] G. Song, Y. Liu, and X. Wang, "Revisiting the sibling head in object detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11563–11572, 2020.

[89] Y. Wu, Y. Chen, L. Yuan, Z. Liu, L. Wang, H. Li, and Y. Fu, "Rethinking classification and localization for object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10186–10195, 2020.

[90] Z. Ge, S. Liu, Z. Li, O. Yoshie, and J. Sun, "Ota: Optimal transport assignment for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 303–312, 2021.

[91] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, *et al.*, "Yolov6: A single-stage object detection framework for industrial applications," *arXiv preprint arXiv:2209.02976*, 2022.

[92] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13733–13742, 2021.

[93] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang, "Tood: Task-aligned one-stage object detection," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3490–3499, IEEE Computer Society, 2021.

[94] H. Zhang, Y. Wang, F. Dayoub, and N. Sunderhauf, "Varifocalnet: An iou-aware dense object detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8514–8523, 2021.

[95] Z. Gevorgyan, "Siou loss: More powerful learning for bounding box regression," *arXiv preprint arXiv:2205.12740*, 2022.

[96] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 658–666, 2019.

[97] X. Ding, H. Chen, X. Zhang, K. Huang, J. Han, and G. Ding, "Re-parameterizing your optimizers rather than architectures," *arXiv preprint arXiv:2205.15242*, 2022.

[98] C. Shu, Y. Liu, J. Gao, Z. Yan, and C. Shen, "Channel-wise knowledge distillation for dense prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5311–5320, 2021.

[99] 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," *arXiv preprint arXiv:2207.02696*, 2022.

[100] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing network design strategies through gradient path analysis," *arXiv preprint arXiv:2211.04800*, 2022.

[101] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

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

[103] Alibaba, "TinyNAS." `https://github.com/alibaba/lightweight-neural-architecture-search`, 2023. Accessed: March 18, 2023.

[104] Z. Tan, J. Wang, X. Sun, M. Lin, H. Li, *et al.*, "Giraffedet: A heavy-neck paradigm for object detection," in *International Conference on Learning Representations*, 2021.

[105] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics." `https://github.com/ultralytics/ultralytics`, 2023. Accessed: February 30, 2023.

[106] X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang, and J. Yang, "Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21002–21012, 2020.

[107] Y. Ma, D. Yu, T. Wu, and H. Wang, "Paddlepaddle: An open-source deep learning platform from industrial practice," *Frontiers of Data and Domputing*, vol. 1, no. 1, pp. 105–115, 2019.

[108] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.

[109] W. Xinlong, Z. Rufeng, K. Tao, L. Lei, and S. Chunhua, "Solov2: Dynamic, faster and stronger," in *Proc. NIPS*, 2020.

[110] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," *Advances in neural information processing systems*, vol. 31, 2018.

[111] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, *et al.*, "Pp-yolov2: A practical object detector," *arXiv preprint arXiv:2104.10419*, 2021.

[112] S. Xu, X. Wang, W. Lv, Q. Chang, C. Cui, K. Deng, G. Wang, Q. Dang, S. Wei, Y. Du, *et al.*, "Pp-yoloe: An evolved version of yolo," *arXiv preprint arXiv:2203.16250*, 2022.

[113] L. Rao, "Treenet: A lightweight one-shot aggregation convolutional network," *arXiv preprint arXiv:2109.12342*, 2021.

[114] R. team, "YOLO-NAS by Deci Achieves State-of-the-Art Performance on Object Detection Using Neural Architecture Search." `https://deci.ai/blog/yolo-nas-object-detection-foundation-model/`, 2023. Accessed: May 12, 2023.

[115] X. Chu, L. Li, and B. Zhang, "Make repvgg greater again: A quantization-aware approach," *arXiv preprint arXiv:2212.01593*, 2022.

[116] S. Shao, Z. Li, T. Zhang, C. Peng, G. Yu, X. Zhang, J. Li, and J. Sun, "Objects365: A large-scale, high-quality dataset for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8430–8439, 2019.