A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 TO YOLOV8 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

April 4, 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. 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.

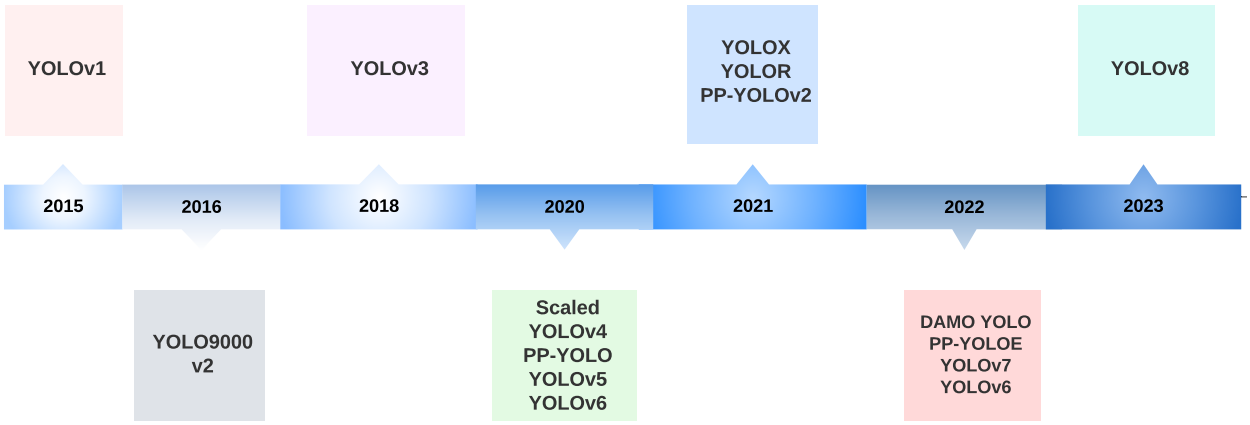

Figure 1: A timeline of YOLO versions.

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 AP. 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.

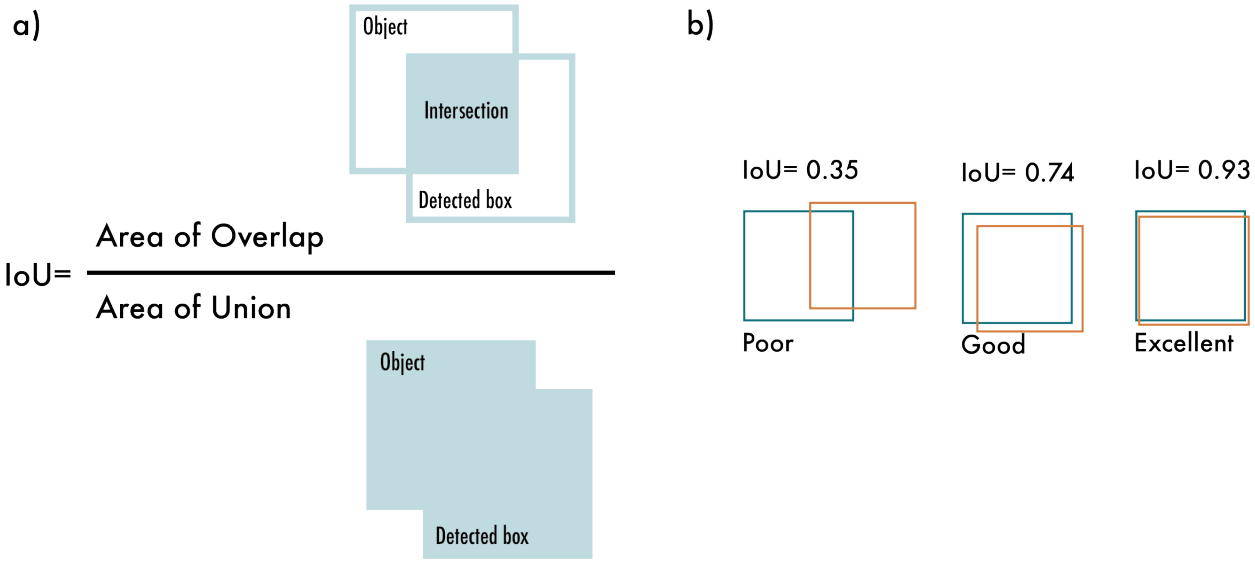

Figure 2: Intersection over Union (IoU). a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; b) examples of three different IoU values for different box locations.

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.

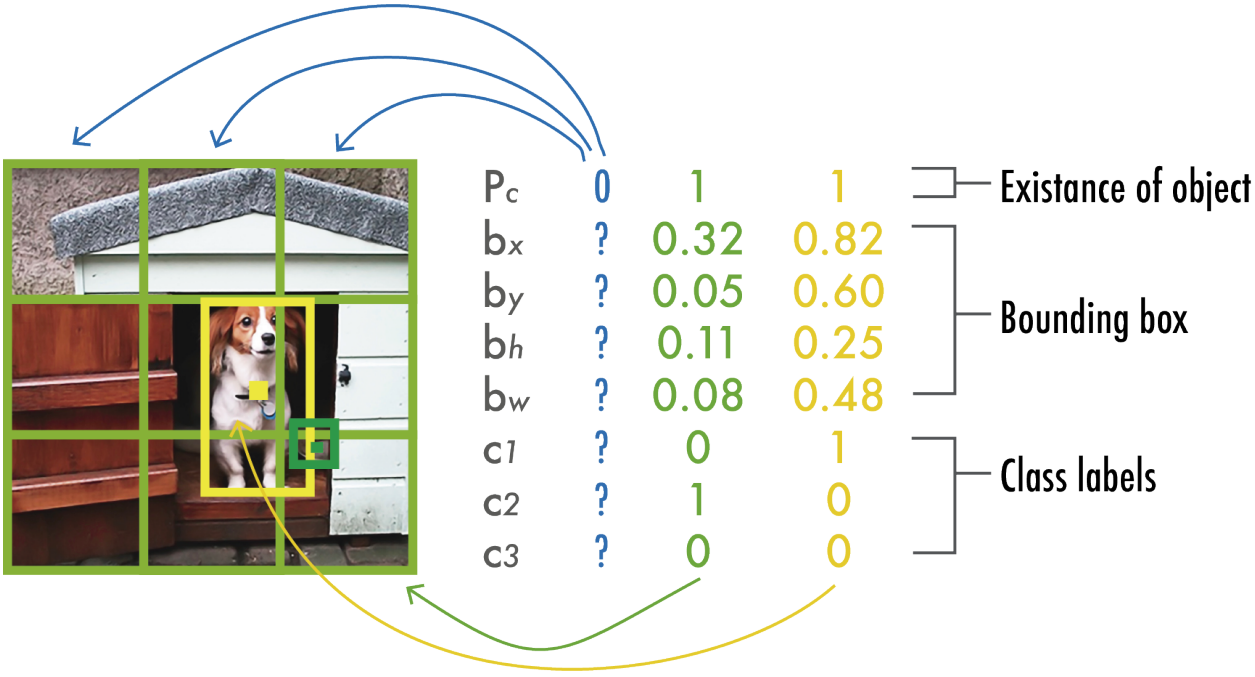

Figure 4: YOLO output prediction. The figure depicts a simplified YOLO model with a three-by-three grid, three classes, and a single class prediction per grid element to produce a vector of eight values.

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

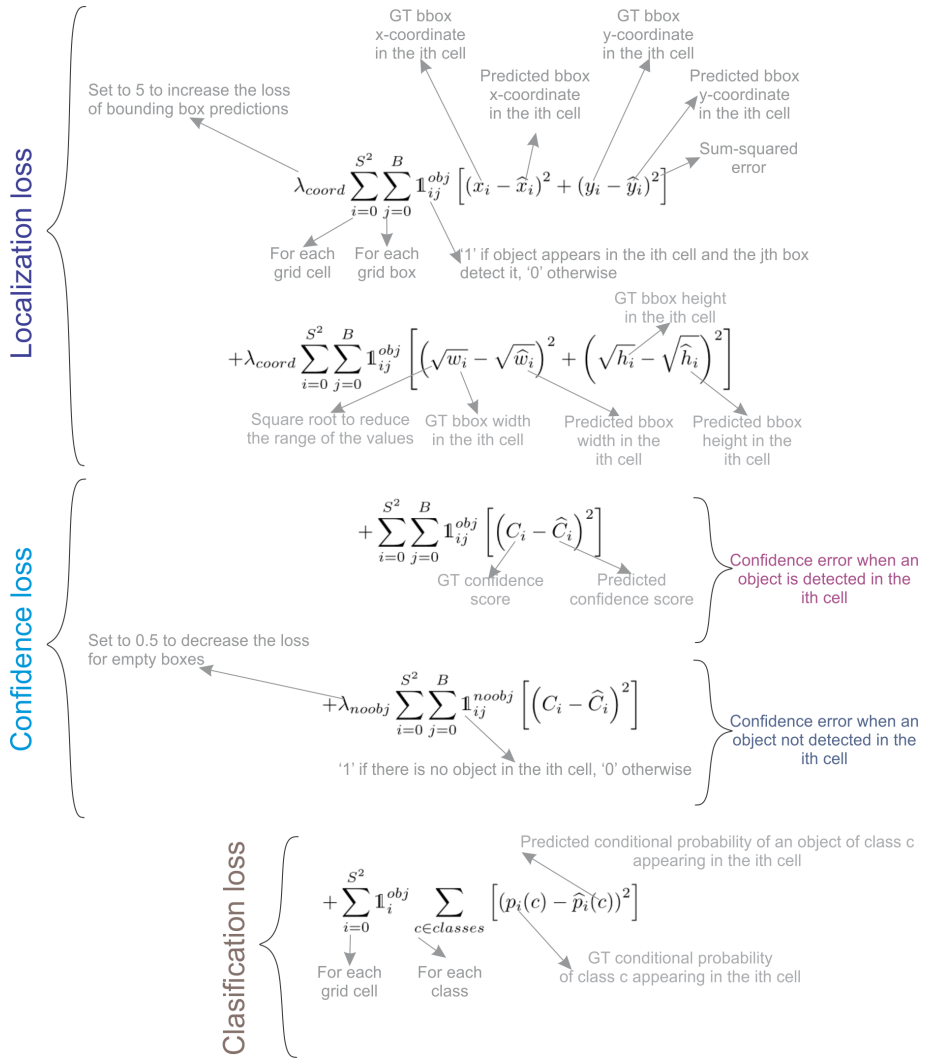

Figure 5: YOLO cost function: includes localization loss for bounding box coordinates, confidence loss for object presence or absence, and classification loss for category prediction accuracy.

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.

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.

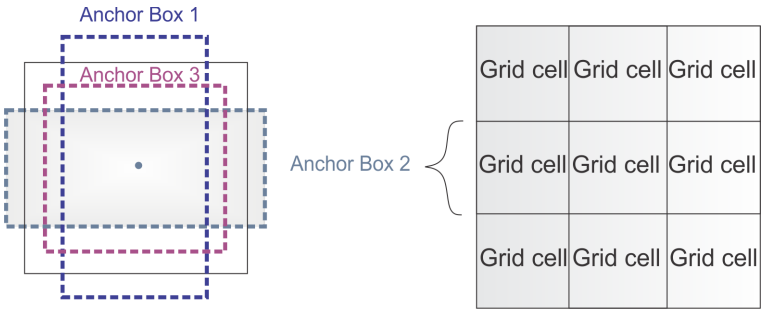

Figure 6: Anchor boxes. YOLOv2 defines multiple anchor boxes for each grid cell.

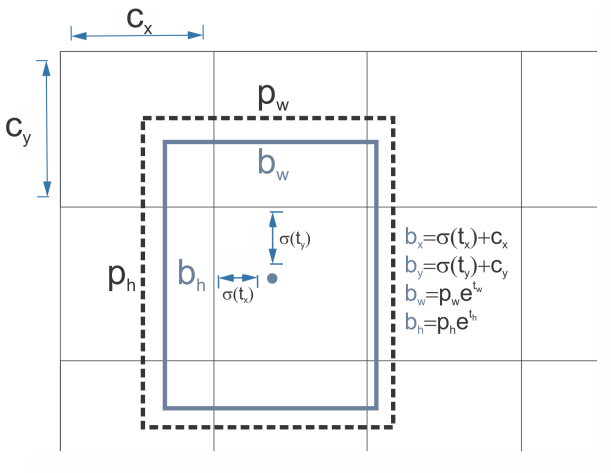

Figure 7: Bounding boxes prediction. The box's center coordinates are obtained with the predicted t_x, t_y values passing through a sigmoid function and offset by the location of the grid cell c_x, c_y. The width and height of the final box use the prior width p_w and height p_h scaled by e^{t_w} and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

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.

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.

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	$13 \times 13 \times 125$

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.