**YOLO Object Detection Explained**

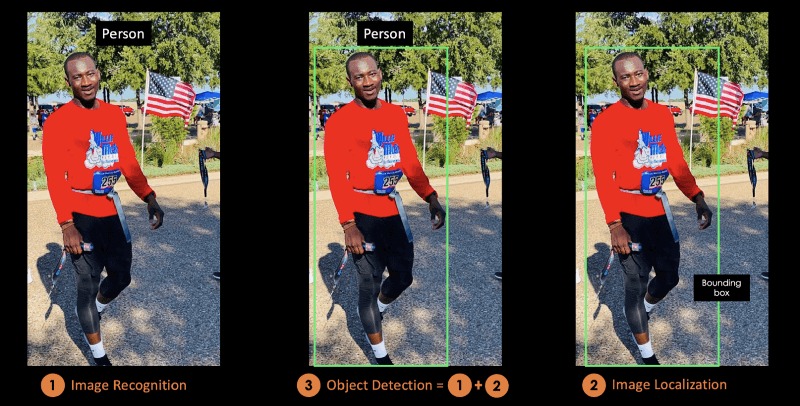
Understand YOLO object detection, its benefits, how it has evolved over the last couple of years and some real-life applications.

Object detection is a technique used in computer vision for the identification and localization of objects within an image or a video.

Image Localization is the process of identifying the correct location of one or multiple objects using bounding boxes, which correspond to rectangular shapes around the objects.

This process is sometimes confused with image classification or image recognition, which aims to predict the class of an image or an object within an image into one of the categories or classes.

The illustration below corresponds to the visual representation of the previous explanation. The object detected within the image is “Person.”



## What is YOLO?

**You Only Look Once (YOLO)** is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in their famous research paper “**You Only Look Once: Unified, Real-Time Object Detection**”.

The authors frame the object detection problem as a regression problem instead of a classification task by spatially separating bounding boxes and associating probabilities to each of the detected images using a single **Convolutional Neural Network (CNN).**

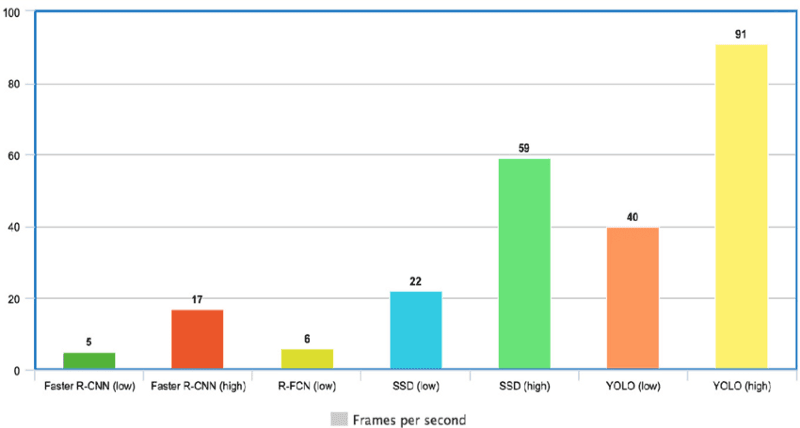
## What Makes YOLO Popular for Object Detection?

Some of the reasons why YOLO is leading the competition include its:

* Speed
* Detection accuracy
* Good generalization
* Open-source

### 1- Speed

YOLO is extremely fast because it does not deal with complex pipelines. It can process images at 45 Frames Per Second (FPS). In addition, YOLO reaches more than twice the mean Average Precision (mAP) compared to other real-time systems, which makes it a great candidate for real-time processing.



**YOLO Speed compared to other state-of-the-art object detectors**

### 2- High detection accuracy

YOLO is far beyond other state-of-the-art models in accuracy with very few background errors.

### 3- Better generalization

This is especially true for the new versions of YOLO, which will be discussed later in the article. With those advancements, YOLO pushed a little further by providing a better generalization for new domains, which makes it great for applications relying on fast and robust object detection.

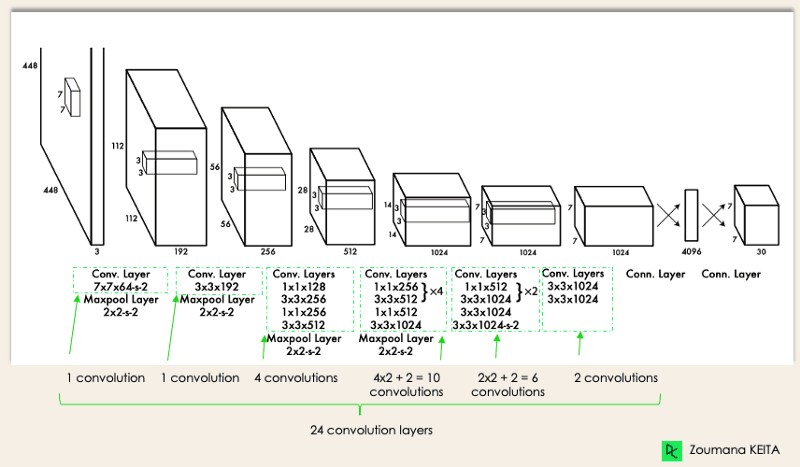
For instance the Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks paper shows that the first version YOLOv1 has the lowest mean average precision for the automatic detection of melanoma disease, compared to YOLOv2 and YOLOv3.

### 4- Open source

Making YOLO open-source led the community to constantly improve the model. This is one of the reasons why YOLO has made so many improvements in such a limited time.

## YOLO Architecture

YOLO architecture is similar to GoogleNet. As illustrated below, it has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.



YOLO Architecture

The architecture works as follows:

* Resizes the input image into 448x448 before going through the convolutional network.
* A 1x1 convolution is first applied to reduce the number of channels, which is then followed by a 3x3 convolution to generate a cuboidal output.
* The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function.
* Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting.

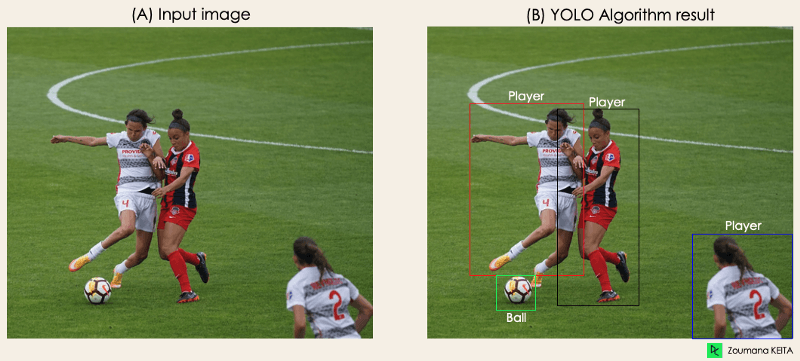
## How Does YOLO Object Detection Work?

Now that you understand the architecture, let’s have a high-level overview of how the YOLO algorithm performs object detection using a simple use case.

*“Imagine you built a YOLO application that detects players and soccer balls from a given image.*

*But how can you explain this process to someone, especially non-initiated people?*

*→ That is the whole point of this section. You will understand the whole process of how YOLO performs object detection; how to get image (B) from image (A)”*

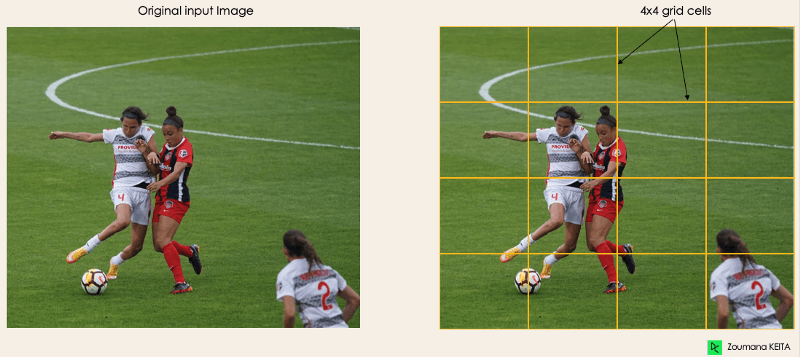


The algorithm works based on the following four approaches:

* Residual blocks
* Bounding box regression
* Intersection Over Unions or IOU for short
* Non-Maximum Suppression.

### 1- Residual blocks

This first step starts by dividing the original image (A) into NxN grid cells of equal shape, where N in our case is 4 shown on the image on the right. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.



### 2- Bounding box regression

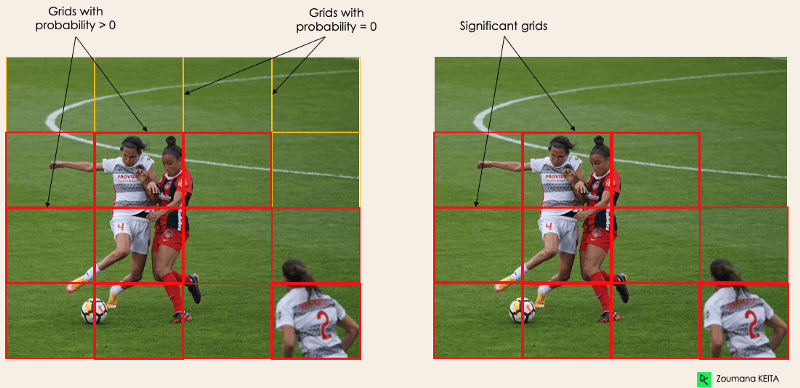
The next step is to determine the bounding boxes which correspond to rectangles highlighting all the objects in the image. We can have as many bounding boxes as there are objects within a given image.

YOLO determines the attributes of these bounding boxes using a single regression module in the following format, where Y is the final vector representation for each bounding box.

Y = [pc, bx, by, bh, bw, c1, c2]

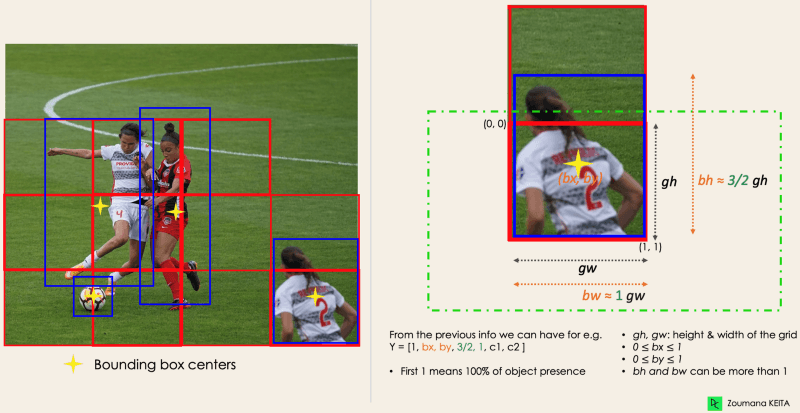
This is especially important during the training phase of the model.

* pc corresponds to the probability score of the grid containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is the simplified version since the probability of each yellow cell is zero (insignificant).



* bx, by are the x and y coordinates of the center of the bounding box with respect to the enveloping grid cell.
* bh, bw correspond to the height and the width of the bounding box with respect to the enveloping grid cell.
* c1 and c2 correspond to the two classes Player and Ball. We can have as many classes as your use case requires.

To understand, let’s pay closer attention to the player on the bottom right.



### 3- Intersection Over Unions or IOU

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. Here is the logic behind it:

* The user defines its IOU selection threshold, which can be, for instance, 0.5.
* Then YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area.
* Finally, it ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.

Below is an illustration of applying the grid selection process to the bottom left object. We can observe that the object originally had two grid candidates, and then only “Grid 2” was selected at the end.

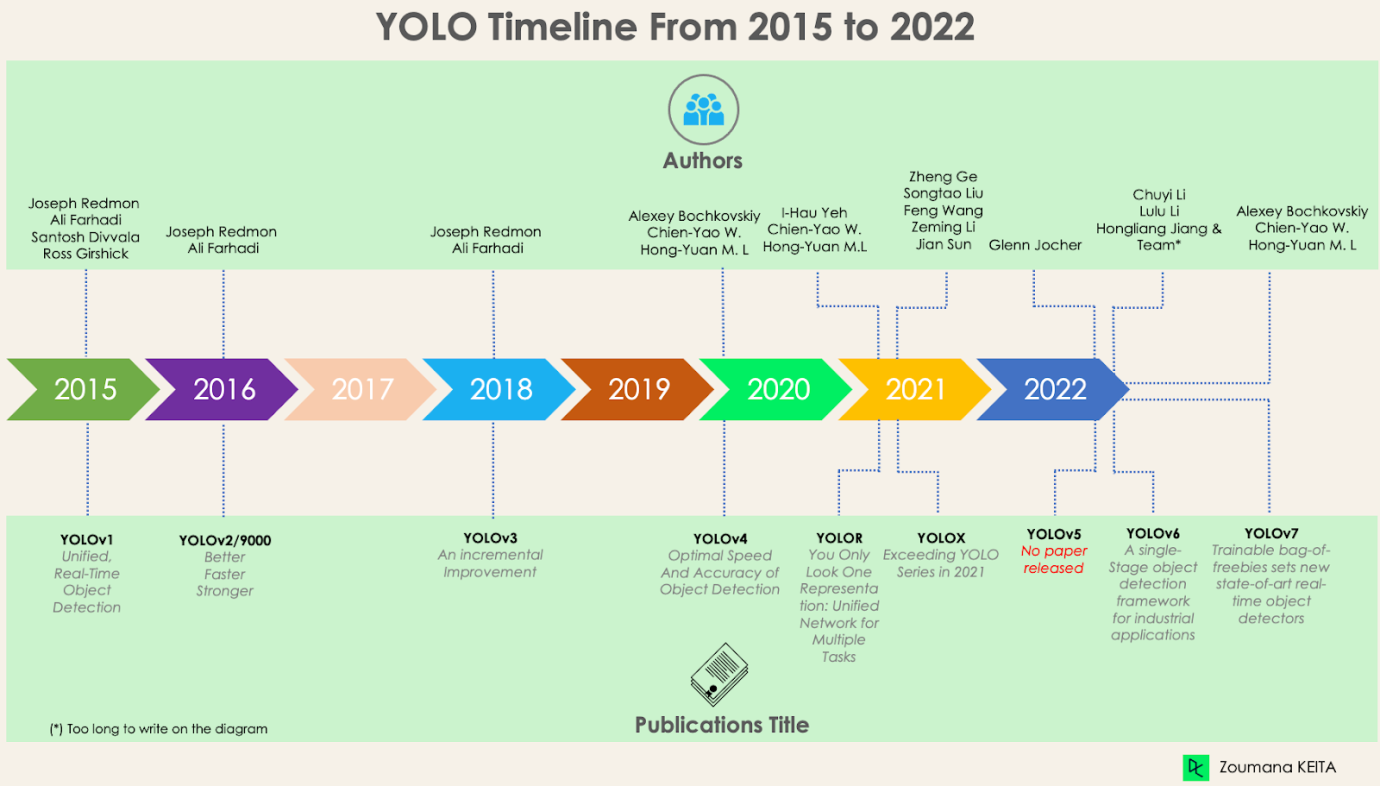


### 4- Non-Max Suppression or NMS

Setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU beyond the threshold, and leaving all those boxes might include noise. Here is where we can use NMS to keep only the boxes with the highest probability score of detection.

## YOLO, YOLOv2, YOLO9000, YOLOv3, YOLOv4, YOLOR, YOLOX, YOLOv5, YOLOv6, YOLOv7 and Differences

Since the first release of YOLO in 2015, it has evolved a lot with different versions. In this section, we will understand the differences between each of these versions.



### YOLO or YOLOv1, the starting point

This first version of YOLO was a game changer for object detection, because of its ability to quickly and efficiently recognize objects.

However, like many other solutions, the first version of YOLO has its own limitations:

* It struggles to detect smaller images within a group of images, such as a group of persons in a stadium. This is because each grid in YOLO architecture is designed for single object detection.
* Then, YOLO is unable to successfully detect new or unusual shapes.
* Finally, the loss function used to approximate the detection performance treats errors the same for both small and large bounding boxes, which in fact creates incorrect localizations.

### YOLOv2 or YOLO9000

YOLOv2 was created in 2016 with the idea of making the YOLO model better, faster and stronger.

The improvement includes but is not limited to the use of [Darknet-19](https://paperswithcode.com/method/darknet-19) as new architecture, batch normalization, higher resolution of inputs, convolution layers with anchors, dimensionality clustering, and (5) Fine-grained features.

#### 1- Batch normalization

Adding a batch normalization layer improved the performance by 2% mAP. This batch normalization included a regularization effect, preventing overfitting.

#### 2- Higher input resolution

YOLOv2 directly uses a higher resolution 448×448 input instead of 224×224, which makes the model adjust its filter to perform better on higher resolution images. This approach increased the accuracy by 4% mAP, after being trained for 10 epochs on the ImageNet data.

#### 3- Convolution layers using anchor boxes

Instead of predicting the exact coordinate of bounding boxes of the objects as YOLOv1 operates, YOLOv2 simplifies the problem by replacing the fully connected layers with anchors boxes. This approach slightly decreased the accuracy, but improved the model recall by 7%, which gives more room for improvement.

#### 4- Dimensionality clustering

The previously mentioned anchor boxes are automatically found by YOLOv2 using k-means dimensionality clustering with k=5 instead of performing a manual selection. This novel approach provided a good trade-off between the recall and the precision of the model.

#### 5- Fine-grained features

YOLOv2 predictions generate 13x13 feature maps, which is of course enough for large object detection. But for much finer objects detection, the architecture can be modified by turning the 26 × 26 × 512 feature map into a 13 × 13 × 2048 feature map, concatenated with the original features. This approach improved the model performance by 1%.

### YOLOv3 — An incremental improvement

[An incremental improvement](https://arxiv.org/pdf/1804.02767.pdf) has been performed on the YOLOv2 to create YOLOv3.

The change mainly includes a new network architecture: [Darknet-53](https://paperswithcode.com/method/darknet-53). This is a 106 neural network, with upsampling networks and residual blocks. It is much bigger, faster, and more accurate compared to [Darknet-19](https://arxiv.org/abs/1612.08242), which is the backbone of YOLOv2. This new architecture has been beneficial on many levels:

#### 1- Better bounding box prediction

A logistic regression model is used by YOLOv3 to predict the objectness score for each bounding box.

#### 2- More accurate class predictions

Instead of using softmax as performed in YOLOv2, independent logistic classifiers have been introduced to accurately predict the class of the bounding boxes. This is even useful when facing more complex domains with overlapping labels (e.g. Person → Soccer Player). Using a softmax would constrain each box to have only one class, which is not always true.

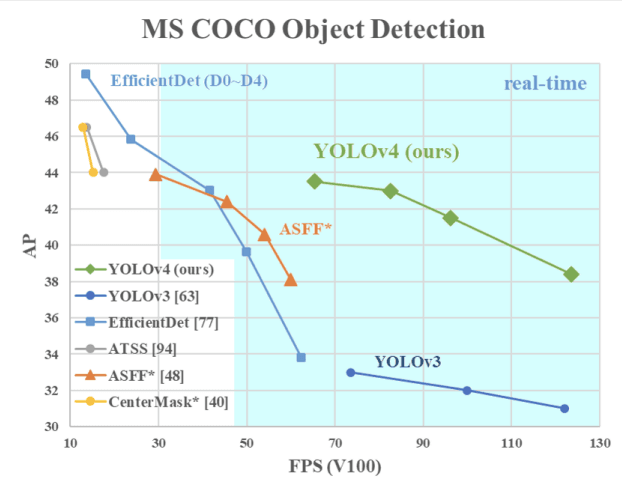
#### 3- More accurate prediction at different scales

YOLOv3 performs three predictions at different scales for each location within the input image to help with the upsampling from the previous layers. This strategy allows getting fine-grained and more meaningful semantic information for a better quality output image.

### YOLOv4 — Optimal Speed and Accuracy of Object Detection

This version of YOLO has an [Optimal Speed and Accuracy of Object Detection](https://arxiv.org/pdf/2004.10934v1.pdf) compared to all the previous versions and other state-of-the-art object detectors.

The image below shows the YOLOv4 outperforming YOLOv3 and FPS in speed by 10% and 12% respectively.



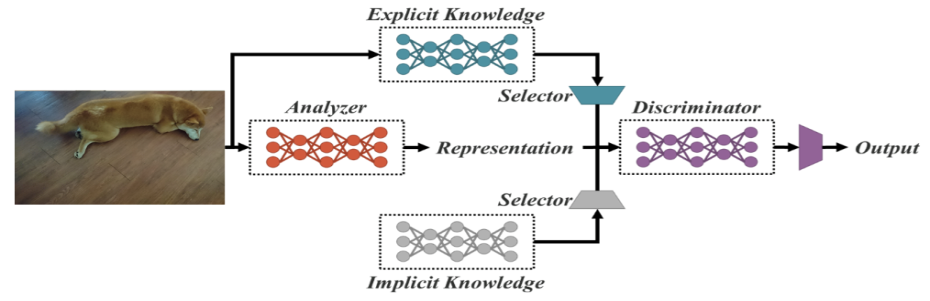
YOLOv4 Speed compared to YOLOv3 and other state-of-the-art object detectors

YOLOv4 is specifically designed for production systems and optimized for parallel computations.

The backbone of YOLOv4’s architecture is [CSPDarknet53](https://paperswithcode.com/method/cspdarknet53), a network containing 29 convolution layers with 3 × 3 filters and approximately 27.6 million parameters.

This architecture, compared to YOLOv3, adds the following information for better object detection:

* Spatial Pyramid Pooling (SPP) block significantly increases the receptive field, segregates the most relevant context features, and does not affect the network speed.
* Instead of the Feature Pyramid Network (FPN) used in YOLOv3, YOLOv4 uses [PANet](https://bio-protocol.org/exchange/minidetail?type=30&id=9907669) for parameter aggregation from different detection levels.
* Data augmentation uses the mosaic technique that combines four training images in addition to a self-adversarial training approach.
* Perform optimal hyper-parameter selection using genetic algorithms.



Unified network architecture

Explicit knowledge is normal or conscious learning. Implicit learning on the other hand is one performed subconsciously (from experience).

Combining these two technics, YOLOR is able to create a more robust architecture based on three processes: (1) feature alignment, (2) prediction alignment for object detection, and (3) canonical representation for multi-task learning

### 1- Prediction alignment

This approach introduces an implicit representation into the feature map of every feature pyramid network (FPN), which improves the precision by about 0.5%.

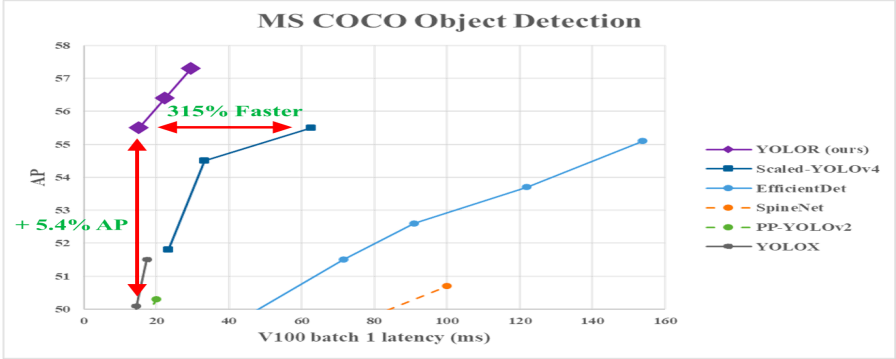
### 2- Prediction refinement for object detection

The model predictions are refined by adding implicit representation to the output layers of the network.

### 3- Canonical representation for multi-task learning

Performing multi-task training requires the execution of the joint optimization on the loss function shared across all the tasks. This process can decrease the overall performance of the model, and this issue can be mitigated with the integration of the canonical representation during the model training.

From the following graphic, we can observe that YOLOR achieved on the [MS COCO](https://viso.ai/deep-learning/yolor/#:~:text=vision.%20On%20the-,MS%20COCO,-dataset%2C%20the%20mean) data state-of-the-art inference speed compared to other models.



YOLOR performance vs. YOLOv4 and other models

## YOLOX — Exceeding YOLO Series in 2021

This uses a baseline that is a modified version of YOLOv3, with Darknet-53 as its backbone.

Published in the paper [Exceeding YOLO Series in 2021](https://arxiv.org/abs/2107.08430), YOLOX brings to the table the following four key characteristics to create a better model compared to the older versions.

### 1- An efficient decoupled head

The coupled head used in the previous YOLO versions is shown to reduce the models’ performance. YOLOX uses a decoupled instead, which allows separating classification and localization tasks, thus increasing the performance of the model.

### 2- Robust data augmentation

Integration of Mosaic and [MixUp](https://paperswithcode.com/method/mixup) into the data augmentation approach considerably increased YOLOX’s performance.

### 3- An anchor-free system

 Anchor-based algorithms perform clustering under the hood, which increases the inference time. Removing the anchor mechanism in YOLOX reduced the number of predictions per image, and significantly improved inference time.

### 4- SimOTA for label assignment

Instead of using the intersection of union (IoU) approach, the author introduced SimOTA, a more robust label assignment strategy that achieves state-of-the-art results by not only reducing the training time but also avoiding extra hyperparameter issues. In addition to that, it improved the detection mAP by 3%.

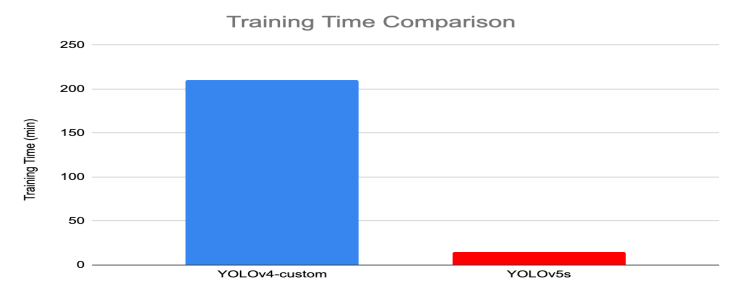
## YOLOv5

YOLOv5, compared to other versions, does not have a published research paper, and it is the first version of YOLO to be implemented in Pytorch, rather than Darknet.

Released by Glenn Jocher in June 2020, YOLOv5, similarly to YOLOv4, uses [CSPDarknet53](https://paperswithcode.com/method/cspdarknet53) as the backbone of its architecture. The release includes five different model sizes: YOLOv5s (smallest), YOLOv5m, YOLOv5l, and YOLOv5x (largest).

One of the major improvements in YOLOv5 architecture is the integration of the [Focus layer](https://github.com/ultralytics/yolov5/discussions/3181m1), represented by a single layer, which is created by replacing the first three layers of YOLOv3. This integration reduced the number of layers, and number of parameters and also increased both forward and backward speed without any major impact on the mAP.

The following illustration compares the training time between YOLOv4 and YOLOv5s.



Training time comparison between YOLOv4 and YOLOv5

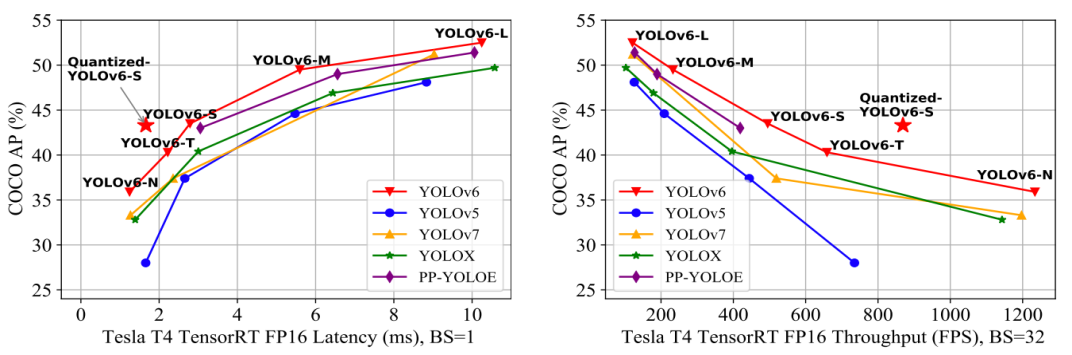
## YOLOv6 — A Single-Stage Object Detection Framework for Industrial Applications

Dedicated to [industrial applications](https://arxiv.org/pdf/2209.02976.pdf) with hardware-friendly efficient design and high performance, the YOLOv6 (MT-YOLOv6) framework was released by [Meituan](https://tech.meituan.com), a Chinese e-commerce company.

Written in Pytorch, this new version was not part of the official YOLO but still got the name YOLOv6 because its backbone was inspired by the original one-stage YOLO architecture.

YOLOv6 introduced three significant improvements to the previous YOLOv5: a hardware-friendly backbone and neck design, an efficient decoupled head, and a more effective training strategy.

YOLOv6 provides outstanding results compared to the previous YOLO versions in terms of accuracy and speed on the COCO dataset as illustrated below.



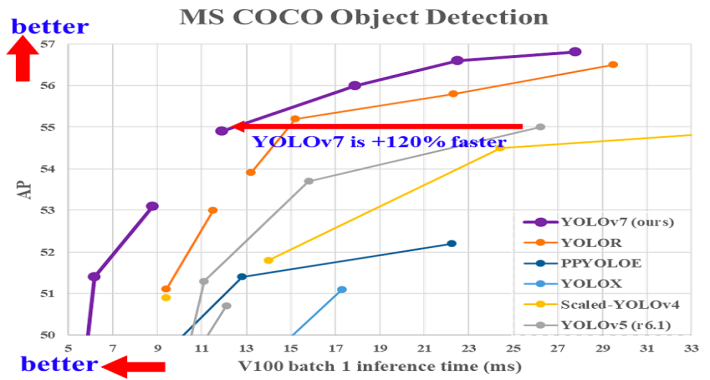
Comparison of state-of-the-art efficient object detectors. All models are tested with TensorRT 7 except that the quantized model is with TensorRT 8

* YOLOv6-N achieved 35.9% AP on the [COCO dataset](https://paperswithcode.com/dataset/coco) at a throughput of 1234 (throughputs) FPS on an NVIDIA Tesla T4 GPU.
* YOLOv6-S reached a new state-of-the-art 43.3% AP at 869 FPS.
* YOLOv6-M and YOLOv6-L also achieved better accuracy performance respectively at 49.5% and 52.3% with the same inference speed.

All these characteristics make YOLOv5, the right algorithm for industrial applications.

## YOLOv7 — Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

YOLOv7 was released in July 2022 in the paper [Trained bag-of-freebies sets new state-of-the-art for real-time object detectors](https://arxiv.org/pdf/2207.02696.pdf). This version is making a significant move in the field of object detection, and it surpassed all the previous models in terms of accuracy and speed.



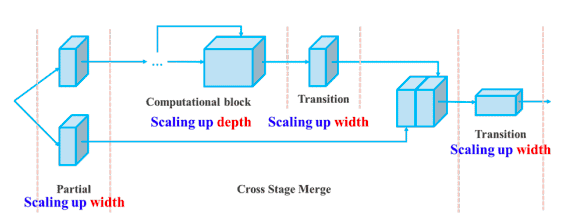
Comparison of YOLOv7 inference time with other real-time object detectors

YOLOv7 has made a major change in its (1) architecture and (2) at the Trainable bag-of-freebies level:

### 1- Architectural level

YOLOv7 reformed its architecture by integrating the Extended Efficient Layer Aggregation Network (E-ELAN) which allows the model to learn more diverse features for better learning.

In addition, YOLOv7 scales its architecture by concatenating the architecture of the models it is derived from such as YOLOv4, Scaled YOLOv4, and YOLO-R. This allows the model to meet the needs of different inference speeds.



Compound scaling up depth and width for concatenation-based model

### 2- Trainable bag-of-freebies

The term **bag-of-freebies** refers to improving the model’s accuracy without increasing the training cost, and this is the reason why YOLOv7 increased not only the inference speed but also the detection accuracy.