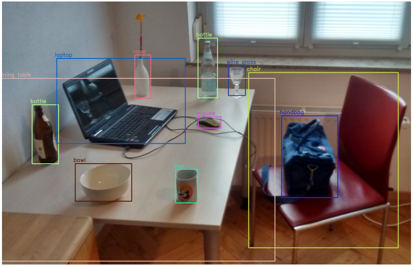
Object detection

Object detection

Similar to segmentation, but we are supposed to return a boundary box containing the object.



- **pro**: no need to strive about borders
- **cons**: multiple outputs of unknown number
  - difficult to train end-to-end
  - no evident loss function

Datasets for object detection

Many datasets for semantic segmentation also provide ground truth for object detection, and viceversa, e.g.

- PASCAL Visual Object Classes
- Coco: a large-scale object detection, segmentation, and captioning dataset composed of over 200K labeled images, spanning 80 categories.

Good aspects of COCO

- **Standardized eval**: mAP@0.5:0.95 is now industry-standard
- **Comparability**: historical benchmarks
- **Strong community tooling**: pycocotools, COCO API, leaderboards
- **Real-world context**: Not toy images; full-scene complexity
- **Multitask**: Object detection, instance/semantic segmentation, keypoints

COCO is the ImageNet of detection.

Recent Datasets

- Objects365
  - 365 categories, over 600k images
  - More comprehensive than COCO
  - Frequently used for training models
- OpenImages
  - over 600 categories
  - Larger scale
  - Hierarchical class structure (has "sub-dog" labels etc.)
- LVIS
  - Long-tail categories (over 1200 fine-grained classes)
  - Addresses COCO's category imbalance
  - Often used for fine-tuning or long-tail evaluation

Long Tail Categories: Head categories occupy most samples whereas tail categories only own a few samples. However, the testing dataset is balanced across categories.

Relevant metrics

- Intersection over Union
- Mean Average Precision

Intersection over Union

Typically, the quality of each individual bounding box is evaluated vs. the corresponding ground truth using **Intersection over Union**

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

A predicted box is considered a **match** (True Positive) if its IoU is higher than a fixed threshold (e.g. 0.5). A set of different thresholds may be considered, e.g. in the range [0.5:0.95] (see next slides)

Mean Average Precision

Mean Average Precision is a metric used in Information Retrieval Systems to evaluate the quality of a list of retrieved entities (documents, detection boxes, etc.)

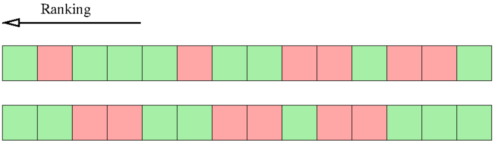
"**Mean**" usually refer to the fact that is averaged over multiple clasess (or also multiple thresholds).

We shall focus on **Average Precision** in next slides.

Average Precision

The problem:

- A retrieval system returns an ordered list of answers (e.g. ordered by relevance, or confidence).
- Some of the answers are correct (True Positives), some are not (False positive).
- How can we compare to different outputs, and provide a sysntetic score for them?



(green = TP, red = FP) which one is better"?

AP definition

Suppose to have N predictions sorted by score. At each prediction i we compute:

- TP_i: cumulative true positives
- FP_i: cumulative false positives

Then,

$$P_i = \frac{TP_i}{TP_i + FP_i} \qquad R_i = \frac{TP_i}{\text{Total Positives}}$$

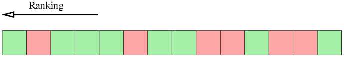
Finally, AP is computed as:

$$AP = \sum_{i=1}^{n} (R_i - R_{i-1}) P_i$$

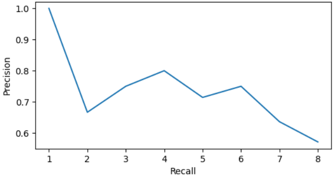
Sometimes (e.g. in COCO) P_i is replaced with an interpolated version ensuring monotonicity.

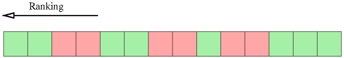
Precision-Recall curve

We can plot Precision in terms of Recall.
Let us see the plot for the two previous retrievals, for a recall passing from $1/8$ to $8/8$.

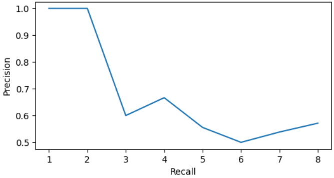


Precisions at different recall positions:
$[1, 2/3, 3/4, 4/5, 5/7, 6/8, 7/11, 8/14]$
MAP $= 0.736$





Precisions at different recall positions:
$[1, 2/2, 3/5, 4/6, 5/9, 6/12, 7/13, 8/14]$
MAP $= 0.679$



Main approaches to Object Detection

- Region Proposal
- Single shots

Deep Object Detection

Two main approaches:

- **Region Proposals** methods (R-CNN, Fast R-CNN, Faster R-CNN). Region Proposals are usually extracted via Selective Search algorithms, aimed to identify possible locations of interest. These algorithms typically exploit the texture and structure of the image, and are object independent.
- **Single shots** methods (Yolo, SSD, Retina-net, FPN). We shall mostly focus on these **really fast** techniques.

Detectron 2

Detectron2 is a pytorch library developed by Facebook AI Research (FAIR) to support rapid implementation and evaluation of novel computer vision research.

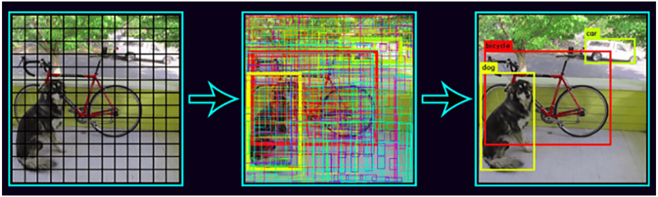
It includes implementations of the following object detection algorithms:

- Mask R-CNN
- RetinaNet
- Faster R-CNN
- RPN
- Efficent Det
- Fast R-CNN
- TensorMask
- PointRend
- DensePose

and more ...

YOLO

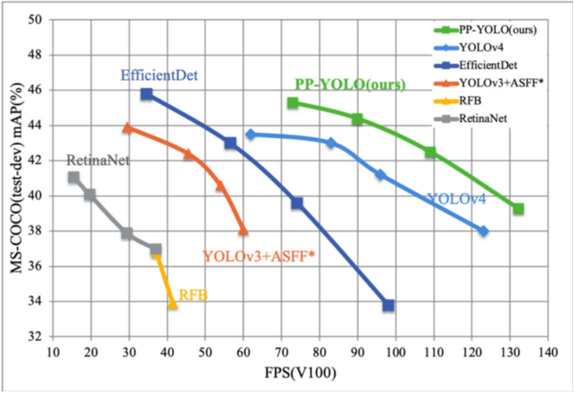
YOLO: Real-Time Object Detection



First release in 2016. One new release every year

> *"I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore."*
>
> *– Joseph Redmon*

Speed and accuracy



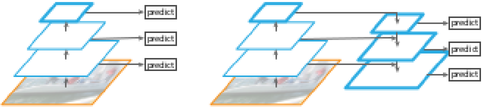
Source PP YOLO repo

YOLOs V8

- Backbone and detection heads
- Predictions format
- YoloV8 loss function
- Multiscale processing
- Non Maximum Suppression

Feature Maps from Backbone + Neck

After the backbone (CSPDarknet) and neck (usually PAN or BiFPN), YOLOv8 produces **multi-scale feature maps** (more on the neck later).



(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

These are usually at 3 different resolutions (e.g., strides of 8, 16, 32). If the input image is 640 × 640, you get:

- P_3 : 80 × 80
- P_4 : 40 × 40
- P_5 : 20 × 20

Each feature map has the usual shape (B,C,H,W).

The Detection Head

Each feature map is passed through a shared detection head that outputs, **for each spatial location** (pixel):

$$(b_x, b_y, b_w, b_h, objectness, c_1, c_2 \dots c_C)$$

where:
- 4 = bounding box prediction (format described below)
- 1 = objectness
- C = number of classes

The final per-pixel prediction vector has shape (one per scale):

$$(B, H, W, 4 + 1 + C) = (B, H, W, N)$$

Bounding Box Format: Distribution Focal Loss (DFL)

YOLOv8 predicts continuous box coordinates but instead of direct regression. It uses DFL, similar to YOLOX/Generalized Focal Loss (GFL).

How DFL works:

- Instead of predicting a real value for width/height (e.g., 83.5 pixels), it predicts a **probability distribution** over discrete bins.
- The center and size (x,y,w,h) of each box are modeled with discrete bins, e.g., 0-16.
- During inference, it takes the expected value (soft argmax) to get a continuous prediction.

This approach improves localization.
Allows better gradient flow than raw L1/IoU box regression.

Model Coordinates with Discrete Bins

In standard object detection, you need to predict continuous dimensions, like e.g. x = 173.8 pixels.

Continuous regression head can be **noisy** to optimize (especially for large boxes), and **prone to over/underestimation**.

Instead, you may discretize the value range, dividing it into a fixed number of bins (e.g. K=16), usually wtih fixed width, and predict a probability distribution over bins using softmax:

$$\hat{p}_0, \hat{p}_1, \ldots \hat{p}_K$$

Continuous Value via Expectation

Instead of choosing the max bin (hard argmax), we compute a continuous value:

$$\hat{x} = \sum_{i=0}^{K-1} i \cdot \hat{p}_i$$

If each bin has width d, then

$$\hat{x}_{final} = d \cdot \hat{x}$$

We are taking the expected value of i based on the distribution.

Why Is This Better?

- Smoother gradients than raw regression.
- More supervision: the loss is not just about the right number, but also about the distribution.
- Works well in noisy cases (e.g., tiny objects or overlapping boxes).
- Avoids overfitting to hard edges of the box.

An example

Let's say you want to predict x = 13.7

You have 16 bins over range [0,16]

The model predicts:

$$\hat{p} = [0.01, 0.02, 0.05, 0.10, 0.50, 0.30, 0.01, \ldots, 0.0]$$

The expected value is:

$$\hat{x} = \sum i \cdot \hat{p}_i = 13.4$$

Pretty close!

Decoding spatial values

Center coordinates and dimensions are interpreted in slightly different ways:

Coordinate	Relative to Grid?	Interpreted as ...
x,y	Yes	Offset from grid cell center
w,h	No	Absolute size (in pixels) or relative to stride

For x and y, the predicted value is an offset Δ, being **added** to the grid cell location and then scaled by stride.

For w and h, it is treated like an absolute length (typically scaled by stride too).

Next argument

YOLO's loss function

Overview of YOLOv8 Training Pipeline

Goal: Learn to predict bounding boxes, objectness, and classes.
Output per location:

$$(x, y, w, h, objectness, c_1, c_2, \ldots, c_C)$$

Key Steps:

- **Forward pass:** predict outputs at every location (multi-scale)
- **Label assignment**: decide which GT boxes belong to which positions
- **Compute loss** (box, obj, class) only where needed
- **Backpropagation**

The heart of localization training is in Step 2: **Dynamic Assignment**

SimOTA-style Dynamic Label Assignment

For each GT box:

- Generates **candidate locations** across feature maps (center or within a radius)
- **Dynamically compute** matching cost for each candidate:

$$cost = \lambda_{cls} \cdot cls_{cost} + \lambda_{loc} \cdot (1 - IoU)$$

- Select top-K lowest cost positions: these are the **positive** locations (K changes depending on how many good matches exist.)

Then:

- **Positives** are trained with box loss + class loss
- **Negatives** are only trained with objectness = 0

These assignments change during training as predictions evolve

Loss Function and Selective Supervision

For selected positive positions:

- Box loss: IoU + DFL (fine-grained localization)
- Class loss, typicaly BCE
- Objectness: BCE (label=1)

For selected negative positions:

- Only objectness: BCE (label = 0)

$$L_{total} = \lambda_{box} \cdot L_{box} + \lambda_{obj} \cdot L_{obj} + \lambda_{cls} \cdot L_{cls}$$

Supervision is selectively applied only where the GT is assigned.

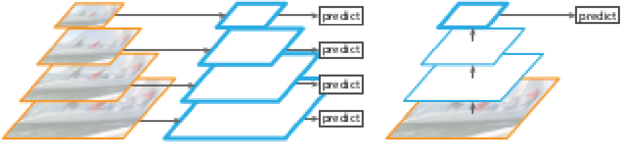
Localization as a Moving Target

- No fixed “owner” of a GT box like in YOLOv1-v3
- Positions compete dynamically to earn responsibility
- Assignment shifts as:
  - Classification confidence improves
  - Box prediction accuracy changes
  - Model learns which parts of the image are more informative

Localization quality is shaped by dynamic competition, not static grids

Multi scale processing

Image Pyramids

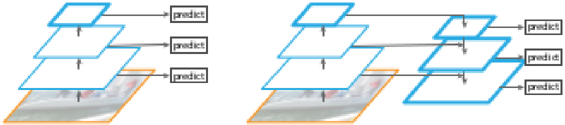


(a) Featurized image pyramid (b) Single feature map

(a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is **slow**.

(b) First systems for fast object detection (like YOLO v1) opted to use only higher level features at the smallest scale. This usually **compromises detection of small objects**.

Feature Pyramids

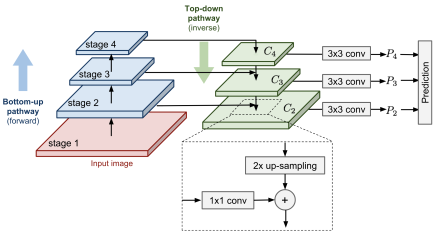


(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

(c) An alternative (Single Shot Detector - SSD) is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a **featurized image pyramid**.

(d) Modern Systems (FPN, RetinaNet, YOLOv3) recombine features along a **backward pathway**. This is as fast as (b) and (c), but more accurate. In the figures, feature maps are indicated by blue outlines and thicker outlines denote **semantically stronger features**.

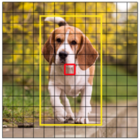
Featurized Image Pyramid

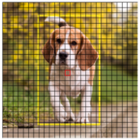


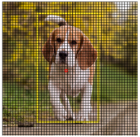
- **Bottom-up pathway** is the normal feedforward computation.
- **Top-down pathway** goes in the inverse direction, adding coarse but semantically stronger feature maps back into the previous pyramid levels of a larger size via lateral connections.
  - First, the higher-level features are spatially upsampled.
  - The feature map coming from the Bottom-up pathway undergoes channels reduction via a 1x1 conv layer
  - Finally, these two feature maps are merged

Non Maximum Suppression

Non Maximum Suppression







YOLOv3 predicts feature maps at scales, eg 20x20, 40x40 and 80x80.

At the end, we have

$$((20 \times 20) + (40 \times 40) + (80 \times 80))x3 = 8400$$

bounding boxes, each one of dimension 85 (4 coordinates, 1 confidence, 80 class probabilities).

How can we reduce this number to the few bounding boxes we expect?

Non Maximum Suppression

These operations are done algorithmically (what a shame)

Essentially they consist in

- **Thresholding by Object Confidence:** first, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold are ignored.
- **Non Maximum Suppression:** NMS addresses the problem of multiple detections of the same image, corresponding to different anchors, adjacent cells in maps.

NMS outline

Divide the bounding boxes BB according to the predicted class c.
Each list BB_c is processed separately

Order BB_c according to the object confidence.

Initialize TruePredictions to an empty list.

while BB_c is not empy:

    pop the first element p from BB_c
    add p to TruePredictions
    remove from BB_c all elements with an IoU with $p > th$

return TruePredictions

Next argument

Evolution and trends

Evolution over time

Version	Year	Key Innovations	Characteristics
YOLOv1	2016	Single CNN for detection	Grid SxS, 2 boxes per cell, direct prediction (no anchors), MSE loss
YOLOv2	2017	Anchor boxes introduced, BatchNorm	Predefined anchors, better mAP, Darknet-19 backbone
YOLOv3	2018	Multiscale prediction (3 scales)	Feature Pyramid Network (FPN), Darknet-53 backbone, BCE loss for classes
YOLOv4	2020	Bag of Freebies + Bag of Specials	Mish activation, PANet neck, CSPDarknet53 backbone, heavy data augmentation
YOLOv5	2020	First PyTorch-native YOLO	PANet neck, anchor-based, lightweight models (n/s/m/l/x), flexible training API
YOLOv6	2022	Efficient design, anchor-based or anchor-free modes	Decoupled head, better deployment on edge devices, industrial focus
YOLOX	2021	Fully anchor-free, SimOTA assignment	Center-based label assignment, strong performance, inspired YOLOv8
YOLOv7	2022	Extendable models, task-agnostic head	Auxiliary head for coarse-to-fine prediction, advanced optimization
YOLOv8	2023	Fully anchor-free, DFL for box regression	Dynamic label assignment (SimOTA-style), simple decoupled head, segmentation and detection tasks unified
YOLO-World	2024	Open-vocabulary detection	Text-conditioned detection (via CLIP), natural language classes

Important trends

- **Anchor-free detection**: starting from YOLOX; mainstream by YOLOv8.
- **Dynamic label assignment**: from static IoU thresholds to flexible cost-based matching (SimOTA, YOLOv8).
- **Better box regression**: Distribution Focal Loss (DFL) replaces direct L1 or smooth L1 losses.
- **Multiscale features**: Present since YOLOv3, refined with better necks (FPN $\rightarrow$ PANet $\rightarrow$ BiFPN).
- **PyTorch adoption**: from YOLOv5 onward, leading to easier deployment and modifications.