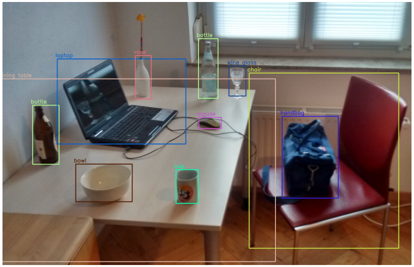
Lesson 13

Object detection

Object detection

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

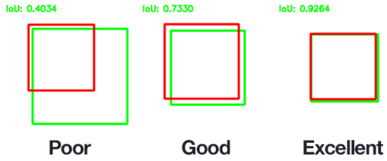


- **pro**: no need to strive about borders
- **cons**: multiple outputs of unknown number
  - difficult to train end-to-end
  - no evident loss function

Intersection over Union

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

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



Then, these must be summed up for all detections, and suitably combined with classification errors.

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.

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.

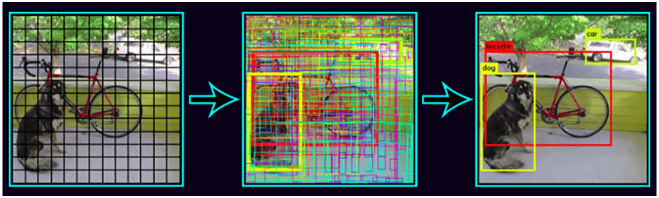
It includes implementations of the following object detection algorithms:

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

and more ...

YOLO

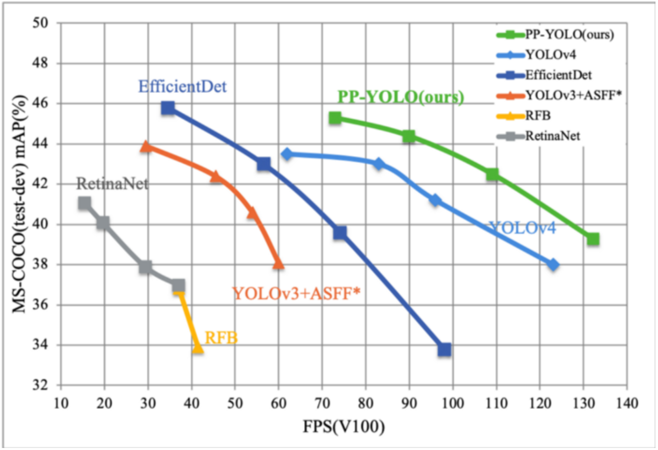
YOLO: Real-Time Object Detection



First release in 2016. Now at version 5.

Suggested reading: YOLO v4 or YOLO v5 or PP-YOLO?

Speed and accuracy



Source PP YOLO repo

Next argument

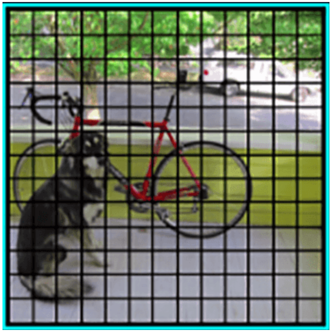
YOLO's architecture

Suggested reading: Yolo-tutorial in pytorch

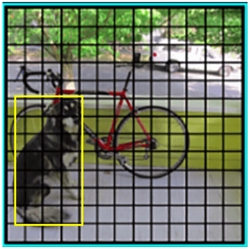
Yolo network

Yolo is a Fully Convolutional Networks. The input is progressively downsampled by a factor $2^5 = 32$.

For instance, an input image of dimension 416x416 would be reduced to a grid of neurons of dimension 13x13: the **feature map**



Who's in charge?



Detection of an object may concern all neurons inside the bounding box.

So, **who's in charge for detection?**

The answer crucially influences the way the network should be trained.

In YOLO, a single neuron is responsible for detection: the one whose grid-cell contains the center of the bounding box.

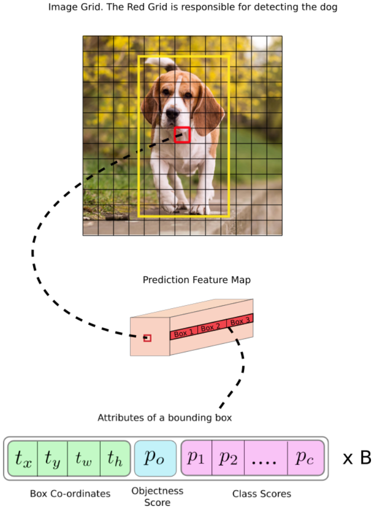
This neuron makes a finite number of **predictions** (e.g. 3).

Predictions

We have 13x13 neurons in the feature map.

Depth-wise, we have (B x (5 + C)) entries, where B represents the number of bounding boxes each cell can predict (say, 3), and C is the number of different object categories.

Each bounding box has 5 + C attributes, which describe the center coordinates (2), the dimensions (2), the objectness score (1) and C class confidences.



Anchor Boxes

Trying to directly predict width and the height of the bounding box leads to **unstable gradients** during training.

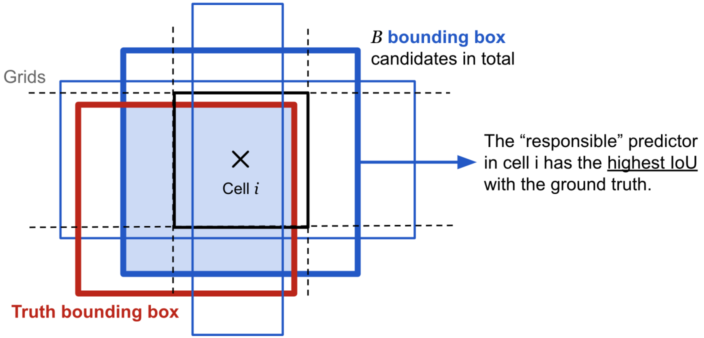
Most of the modern object detectors predict log-space affine transforms for **pre-defined** default bounding boxes called **anchors**.

Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in prediction of three bounding boxes per cell.

The bounding box responsible for detecting the object is one whose anchor has the highest IoU with the ground truth box.

Anchor Boxes

Anchors are choosed by K-means clustering on the training set, reflecting the most likely shapes of bounding boxes.


B bounding box candidates in total
Grids
Cell i
The "responsible" predictor in cell i has the highest IoU with the ground truth.
Truth bounding box

Picture from Object Detection Part 4: Fast Detection Models

Making predictions

We get bounding box predictions from the network outputs t_x, t_y, t_w, t_h relative to the cell at coordinates c_x, c_y for an anchor of dimensions p_w, p_h according to the following formula:

$$\begin{aligned}
b_x &= c_x + \sigma(t_x) \\
b_y &= c_y + \sigma(t_y) \\
b_w &= p_w * e^{t_w} \\
b_h &= p_h * e^{t_h}
\end{aligned}$$

Center Coordinates

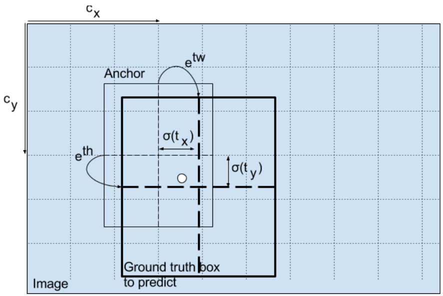
YOLO does not predict the absolute coordinates of the bounding box's center, but offsets which are:

- relative to the top left corner of the grid cell which is predicting the object
- normalised by the dimensions of the cell from the feature map, which is, 1 (that motivates the use of the sigmoid)

For example, consider the case of our dog image. The red cell has coordinates $c_x = 6, c_y = 6$.
If the prediction for center is (0.4, 0.7), this means that the center lies at $b_x = 6.4, b_y = 6.7$ on the 13 x 13 feature map.

Dimensions of the Bounding Box

The dimensions of the bounding box are predicted by applying a log-space transform to the output and then multiplying with the dimensions of the anchor.



The resultant predictions, b_w and b_h, are normalised by the height and width of the image: training labels are chosen this way.

Picture from understanding YOLO

Objectness Score

Objectness score represents the probability that an object is contained inside a bounding box.

The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability.

Class Confidences

Class confidences represent the probabilities of the detected object belonging to a particular class. Before v3, YOLO class scores were computed via a softmax.

Since YOLOv3, multiple sigmoid functions are used instead, considering that objects may belong to multiple (hierachical) categories, and hence labels are not guaranteed to be mutually exclusive.

Next argument

YOLO's loss function

YOLO's loss function

The loss consists of two parts, the **localization loss** for bounding box offset prediction and the **classification loss** for conditional class probabilities.

As usual, we shall use v and $\hat{v}$ to denote a *true* value, and the corresponding *predicted* one.

The lolization loss is

$$\mathcal{L}_{loc} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

where i ranges over cells, and j over bounding boxes.

1_{ij}^{obj} is a delta function indicating whether the j-th bounding box of the cell i is responsible for the object prediction.

YOLO's loss function (2)

The classification loss is the sum of two components, relative to the objectness confidence and the actual classification:

$$\mathcal{L}_{cls} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} (1_{ij}^{obj} + \lambda_{noobj}(1 - 1_{ij}^{obj}))(C_{ij} - \hat{C}_{ij})^2$$

$$+ \sum_{i=0}^{S^2} \sum_{c \in C} 1_i^{obj} (p_i(c) - \hat{p}_i(c))^2$$

λ_{noobj} is a configurable parameter meant to down-weigth the loss contributed by "background" cells containing no objects.
This is important because they are a large majority.

YOLO's loss function (3)

The whole loss is:

$$\mathcal{L} = \lambda_{coord}\mathcal{L}_{loc} + \mathcal{L}_{cls}$$

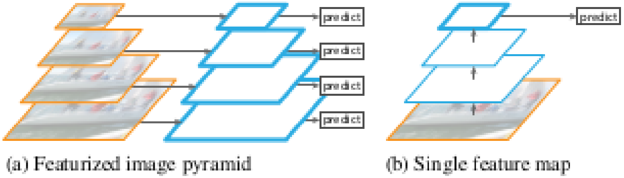
λ_{coord} is an additional parameter, balancing the contribution between $\mathcal{L}_{loc}$ and $\mathcal{L}_{cls}$.

In YOLO, $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$.

Next argument

Multi scale processing

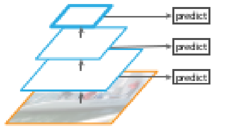
Image Pyramids



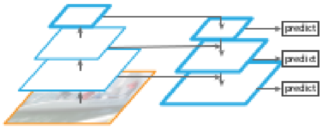
(a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is **slow**.

(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

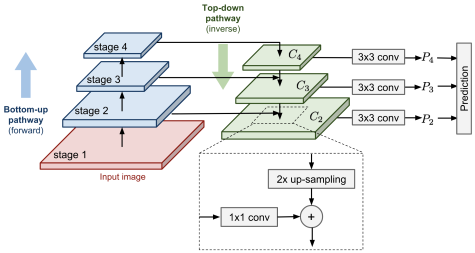


(d) Feature Pyramid Network

(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

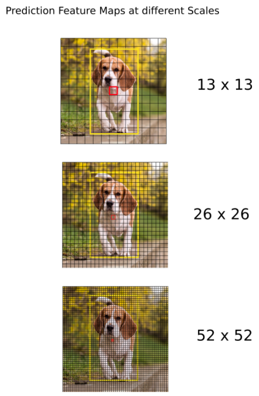


- **Bottom-up pathway** is the normal feedforward computation.
- **Top-down pathway** goes in the inverse direction, adding coarse but semantically stronger feature maps back into the previous pyramid levels of a larger size via lateral connections.
  - First, the higher-level features are spatially 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 (by element-wise addition, or concatenation).

Next argument

Non Maximum Suppression

Non Maximum Suppression



YOLOv3 predicts feature maps at scales 13, 26 and 52.

At the end, we have

$$((13 \times 13) + (26 \times 26) + (52 \times 52))x3 = 10647$$

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

About Ablation

Ablation

Reading articles about Object Detection it is easy to meet discussions relative to **ablative studies**.

So, what is it about?

In general, ablation is removal or destruction of material from an object.

In the context of deep learning, it consists in a progressive removal of components of the network, aimed to asses their contribution to the overall behaviour.

Ablation

Reading articles about Object Detection it is easy to meet discussions relative to **ablative studies**.

So, what is it about?

In general, ablation is removal or destruction of material from an object.

In the context of deep learning, it consists in a progressive removal of components of the network, aimed to asses their contribution to the overall behaviour.

The ablation methodology

Ablation studies are crucial for deep learning research – can't stress this enough. Understanding causality in your system is the most straightforward way to generate reliable knowledge (the goal of any research). And ablation is a very low-effort way to look into causality.

If you take any complicated deep learning experimental setup, chances are you can remove a few modules (or replace some trained features with random ones) with no loss of performance. Get rid of the noise in the research process: do ablation studies.

Can't fully understand your system? Many moving parts? Want to make sure the reason it's working is really related to your hypothesis? Try removing stuff. Spend at least 10% of your experimentation time on an honest effort to disprove your thesis.

Francois Chollet