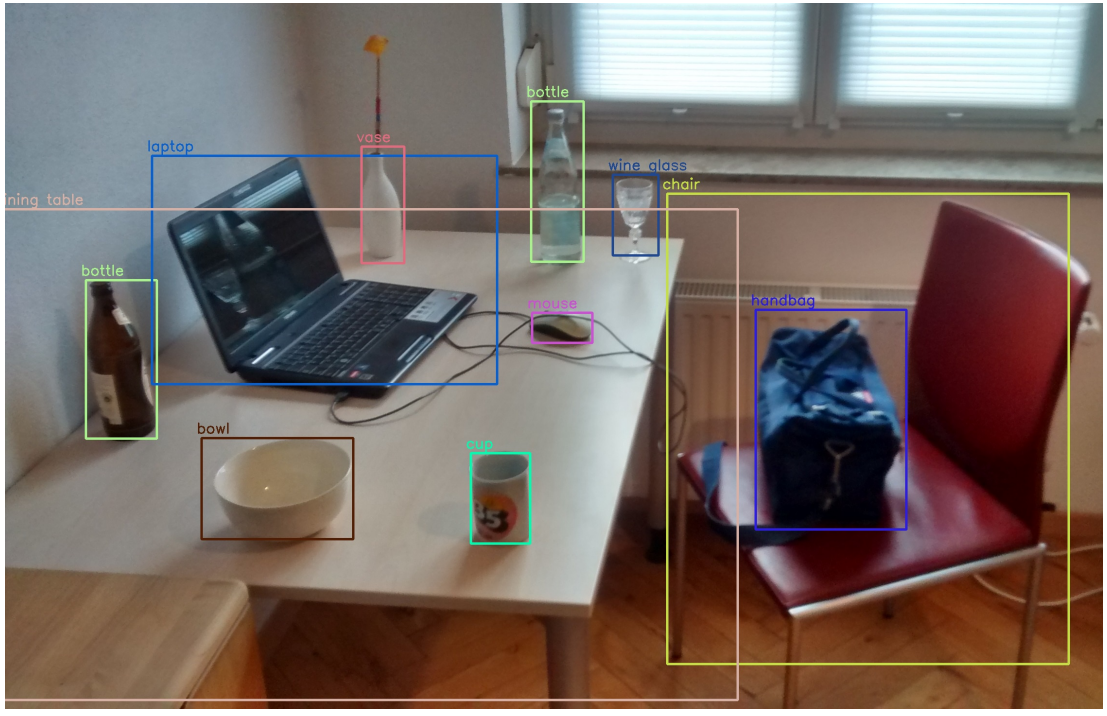
Object Detection & Segmentation

Object detection

Identify the class and **position** of objects in an image, typically returning a boundary box for each prediction.



Suggested reading: Object detection algorithms

Semantic Segmentation

Classify **each pixel** in an image according to the object category it belongs to.

Suggested reading: A short guide to semantic segmentation

Both object detection and semantics segmentation are basic techniques to *understand the content* on an image.

Datasets

Building supervised training sets is expensive, since it requires a complex human operation to create ground truth annotations.

Many datasets provide annotations both for detection and segmentation tasks.

Pascal VOC 2012

PASCAL Visual Object Classes

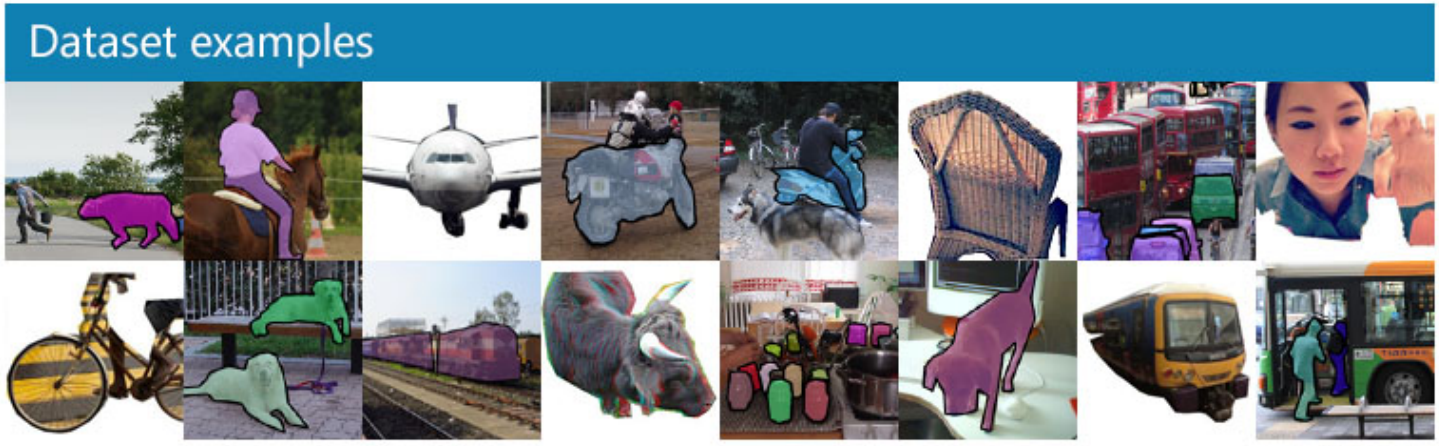
20 classes:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

The train/val data has 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.

Coco dataset

Coco: a large-scale object detection, segmentation, and captioning dataset composed of over 200K labeled images, spanning 80 categories.
There is an annual competiton



Cityscapes dataset

Semantic understanding of urban street scenarios

A large-scale dataset containing stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5K frames, and a larger set of 20K weakly annotated frames.

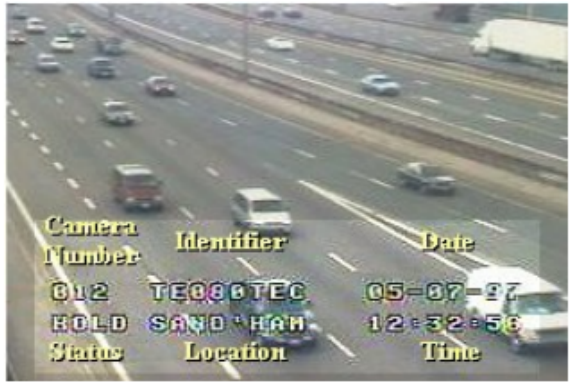
More datasets

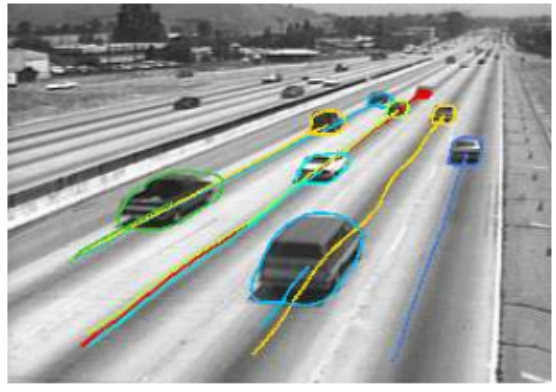
- Syntia a collection of photo-realistic frames rendered from a **virtual** city and **precise** pixel-level semantic annotations for 13 classes: misc, sky, building, road, sidewalk, fence, vegetation, pole, car, sign, pedestrian, cyclist, lane-marking.
- ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. ScanNet is an RGB-D video dataset containing 2.5 million views in more than 1500 scans, annotated with 3D camera poses, surface reconstructions, and instance-level semantic segmentations.
- Sun RGB-D: 10,000 RGB-D images of densely annotated indoor scenes, includes 146617 2D polygons and 58657 3D bounding boxes with accurate object orientations, as well as a 3D room layout and category for scenes.

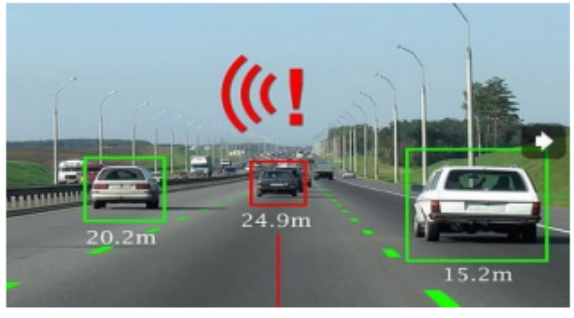
Some applications

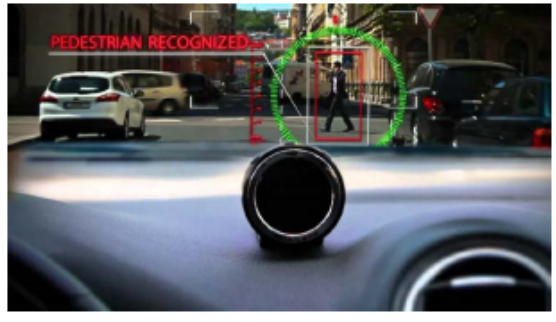
Self driving cars

Tracking cars and pedestrians.

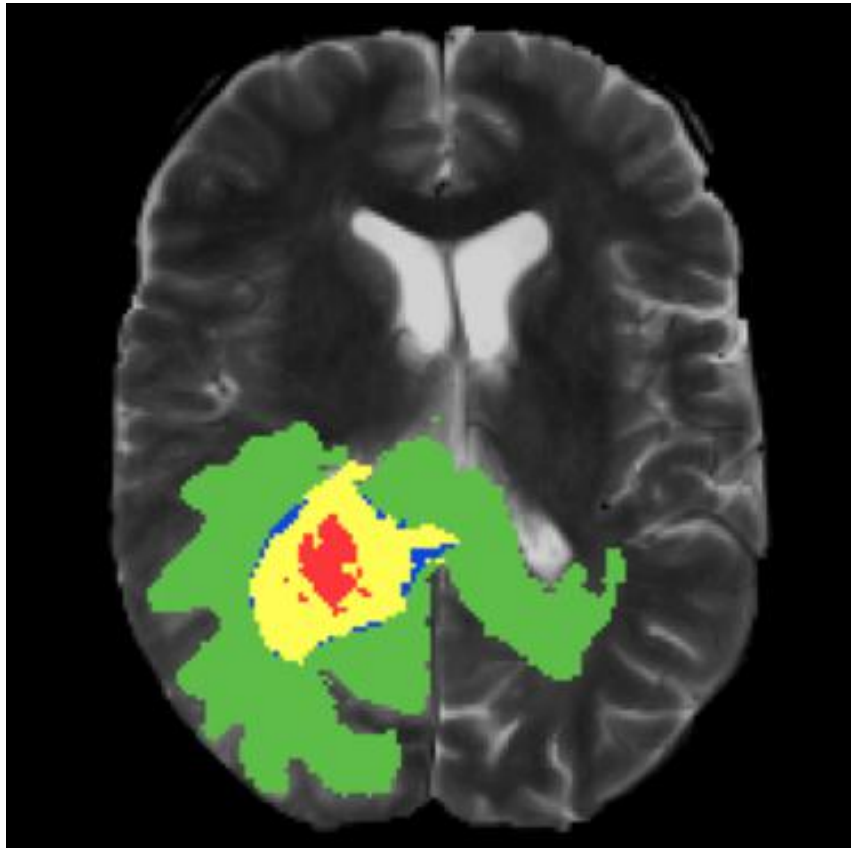








Medical imaging



Video surveillance

Activity recognition and pose estimation

Suggested reading: Keypoint Detection with Transfer Learning

State of the art technologies

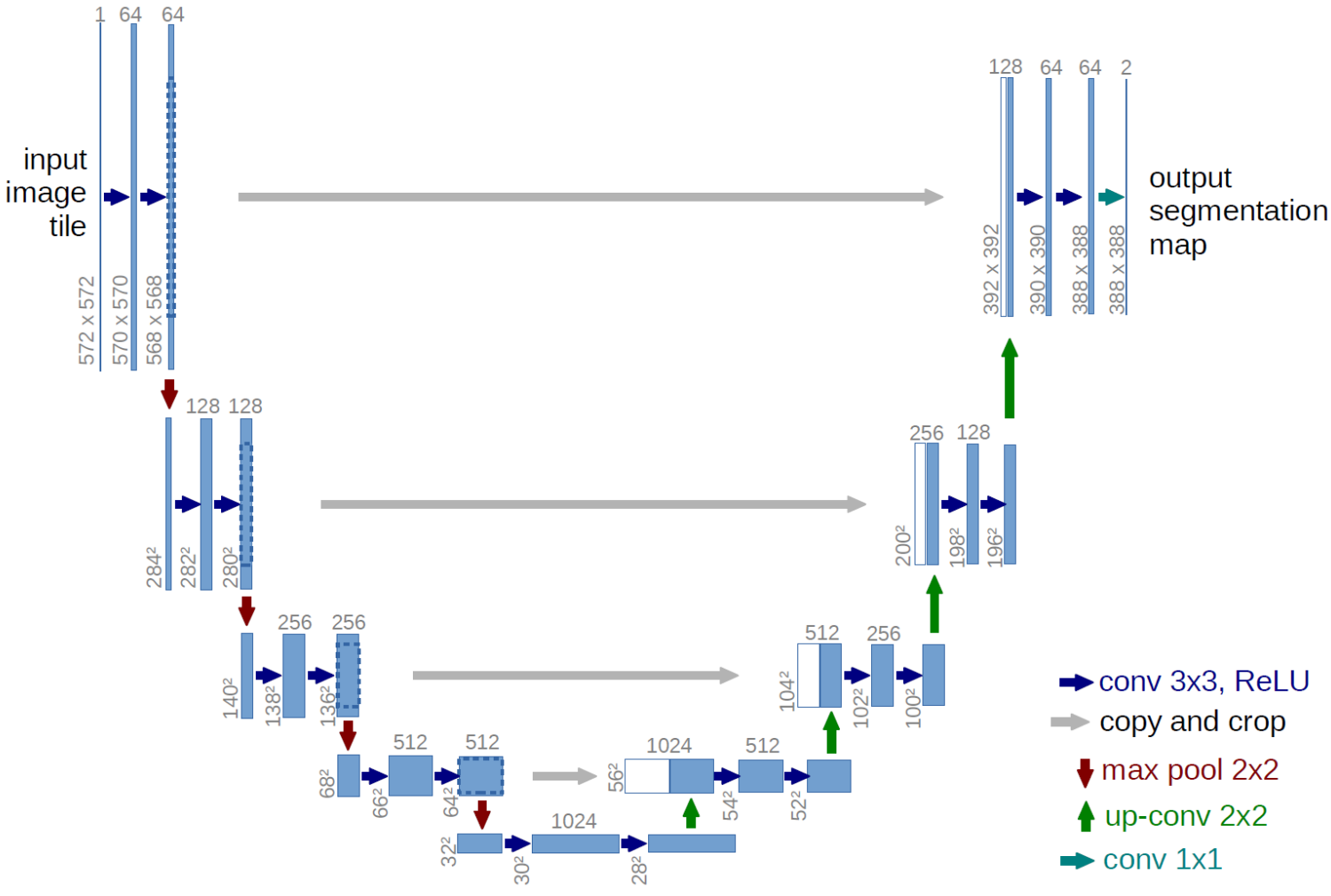
- U-net
- Detectron 2
- Yolo

U-net

Suggested reading:

U-Net: Convolutional Networks for Biomedical Image Segmentation. By O.Ronneberger, P.Fischer, T.Brox

U-net

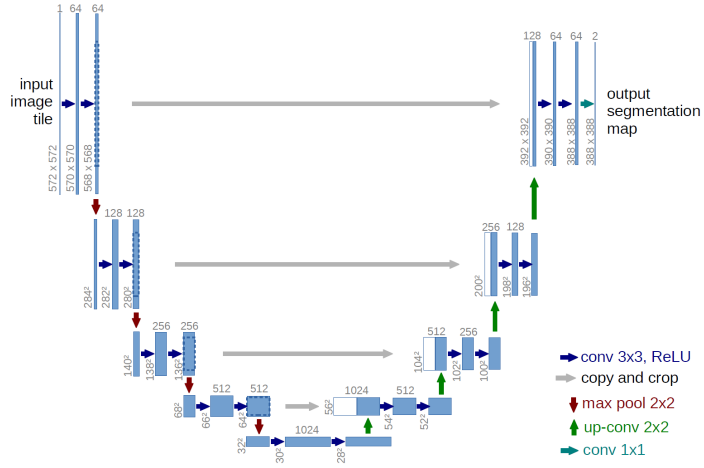

input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-net key properties

It does not rely on a classification network.

U-net learns segmentation in an end-to-end setting.

It can work with relatively few training images (data augmentation was largely exploited) and yields accurate precise segmentations.



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.

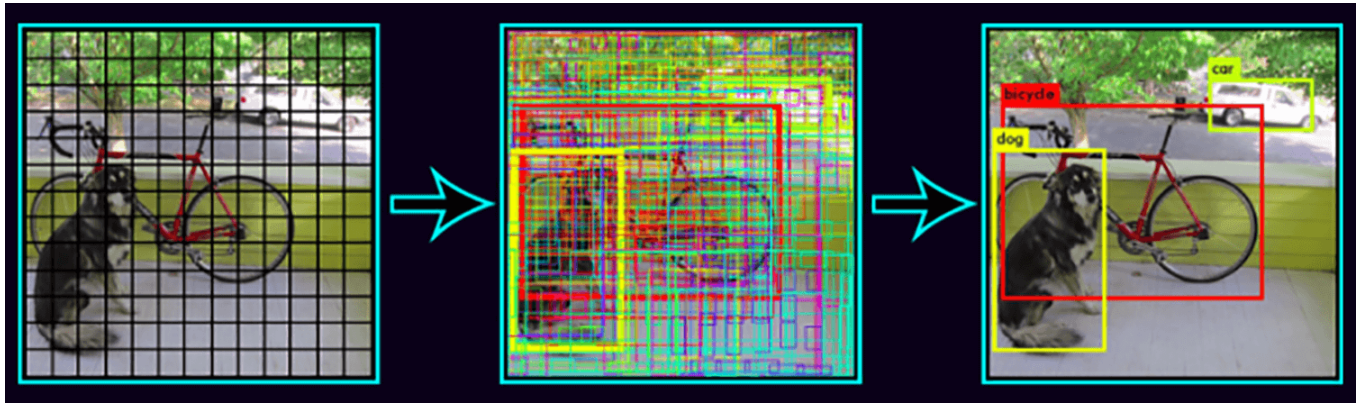
It includes implementations of the following object detection algorithms:

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

and more ...

YOLO

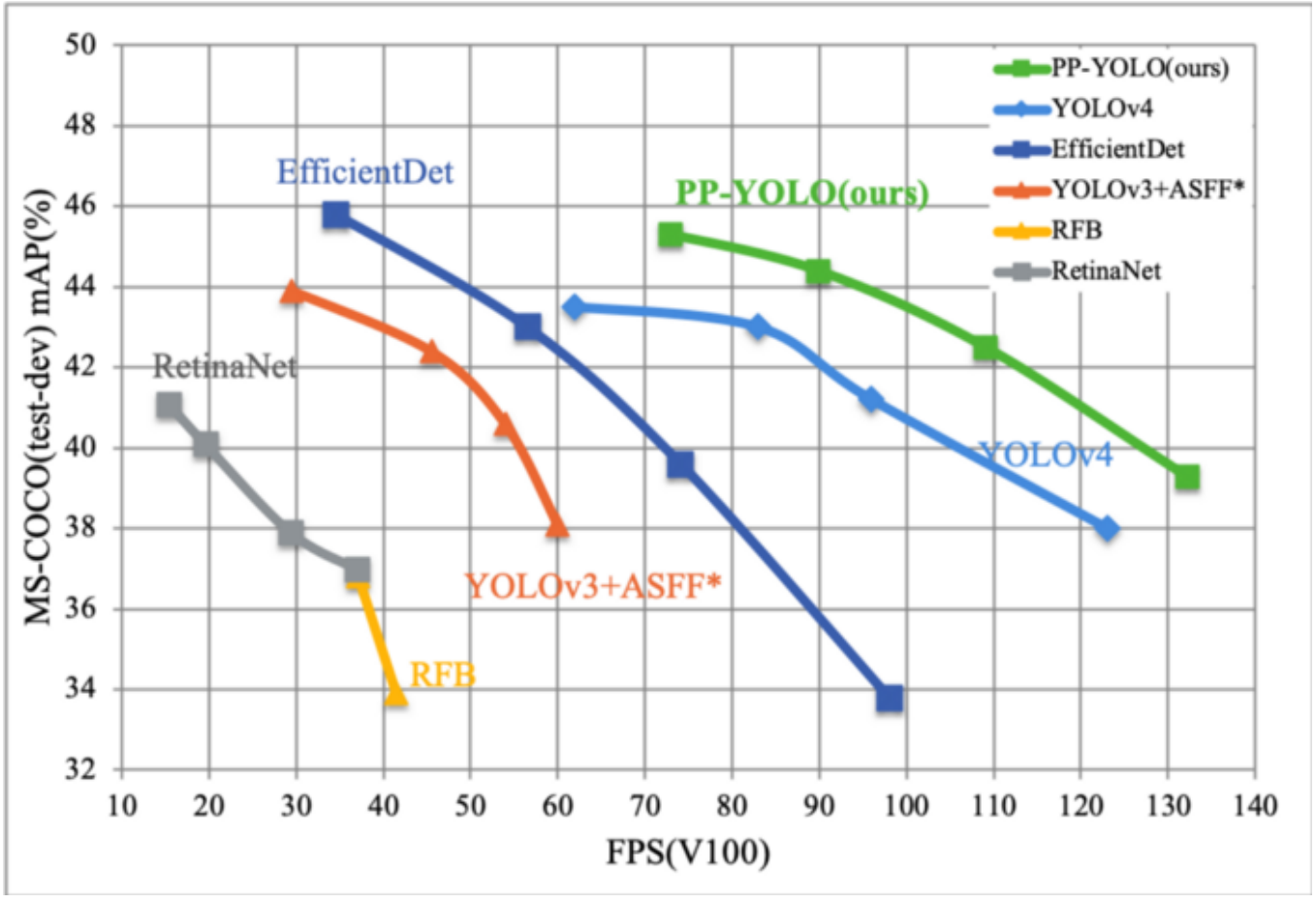
YOLO: Real-Time Object Detection



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

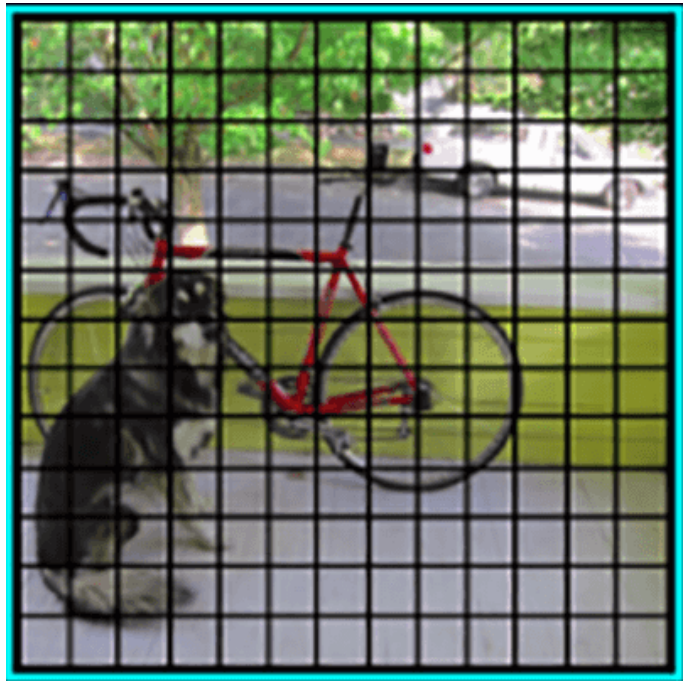
YOLO's architecture

Suggested reading: Yolo-tutorial in pytorch

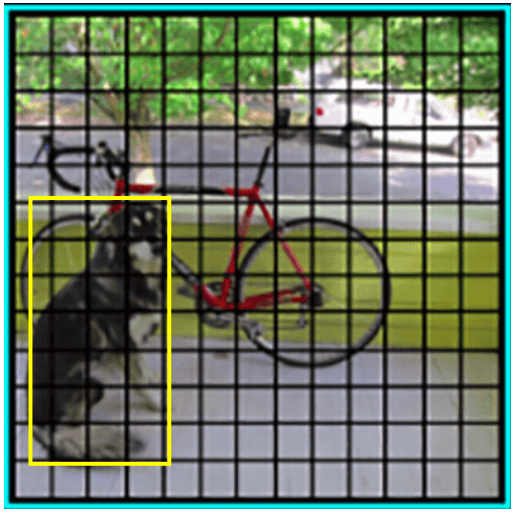
Yolo network

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**



Which neuron is in charge of detection?



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.

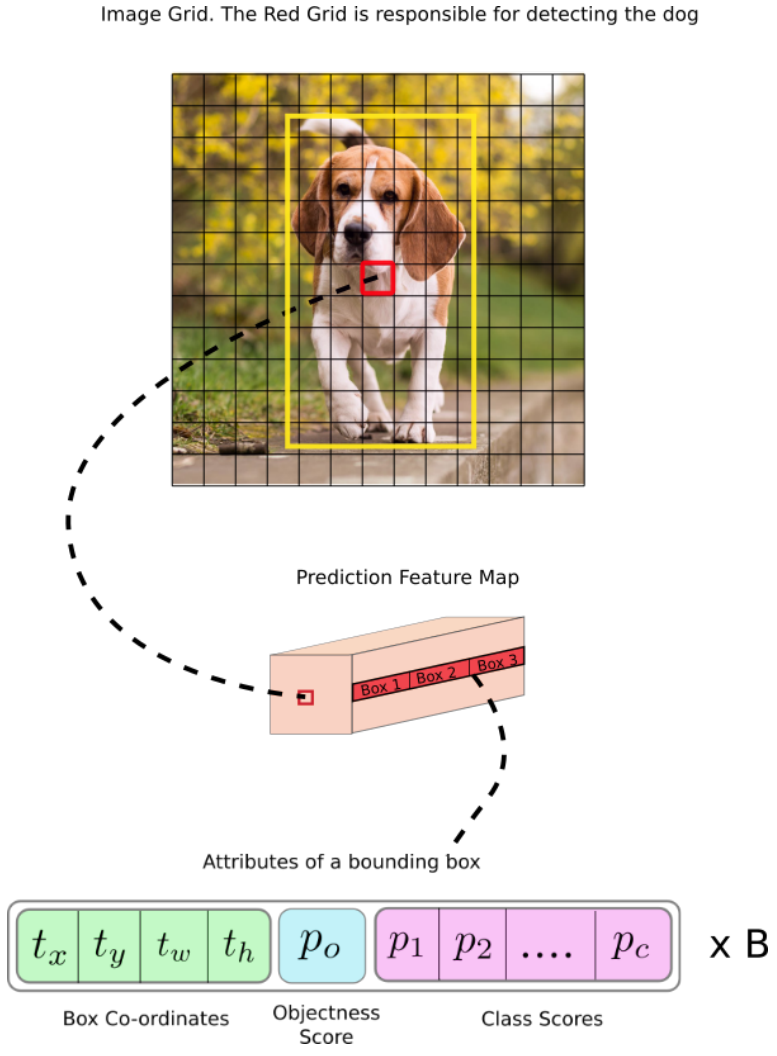
This neuron makes a finite number of **predictions** (e.g. 3).

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 co-ordinates (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.

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.

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:

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

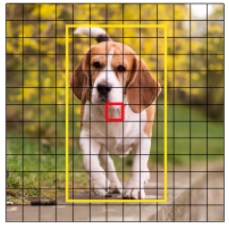
λ_{coord} is an additional parameter, balancing the contribution between $\mathcal{L}_{loc}$ and $\mathcal{L}_{cls}$.

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

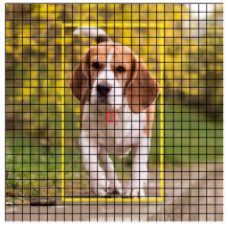
Non Maximum Suppression

Non Maximum Suppression

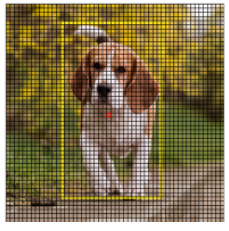
Prediction Feature Maps at different Scales



13 x 13



26 x 26



52 x 52

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

At the end, we have

$$((13\times13)+(26\times26)+(52\times52))x3 = 10647$$

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?

Intersection over Union

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

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

IoU is also used to evalute the classifier, summing the accuracy of all detections, suitably combined with classification errors, the so called **Mean Average Precision** (MAP).

Non Maximum Suppression

The selection of the "best" bounding boxes is done algorithmically, and it usually composed of two phases:

- **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