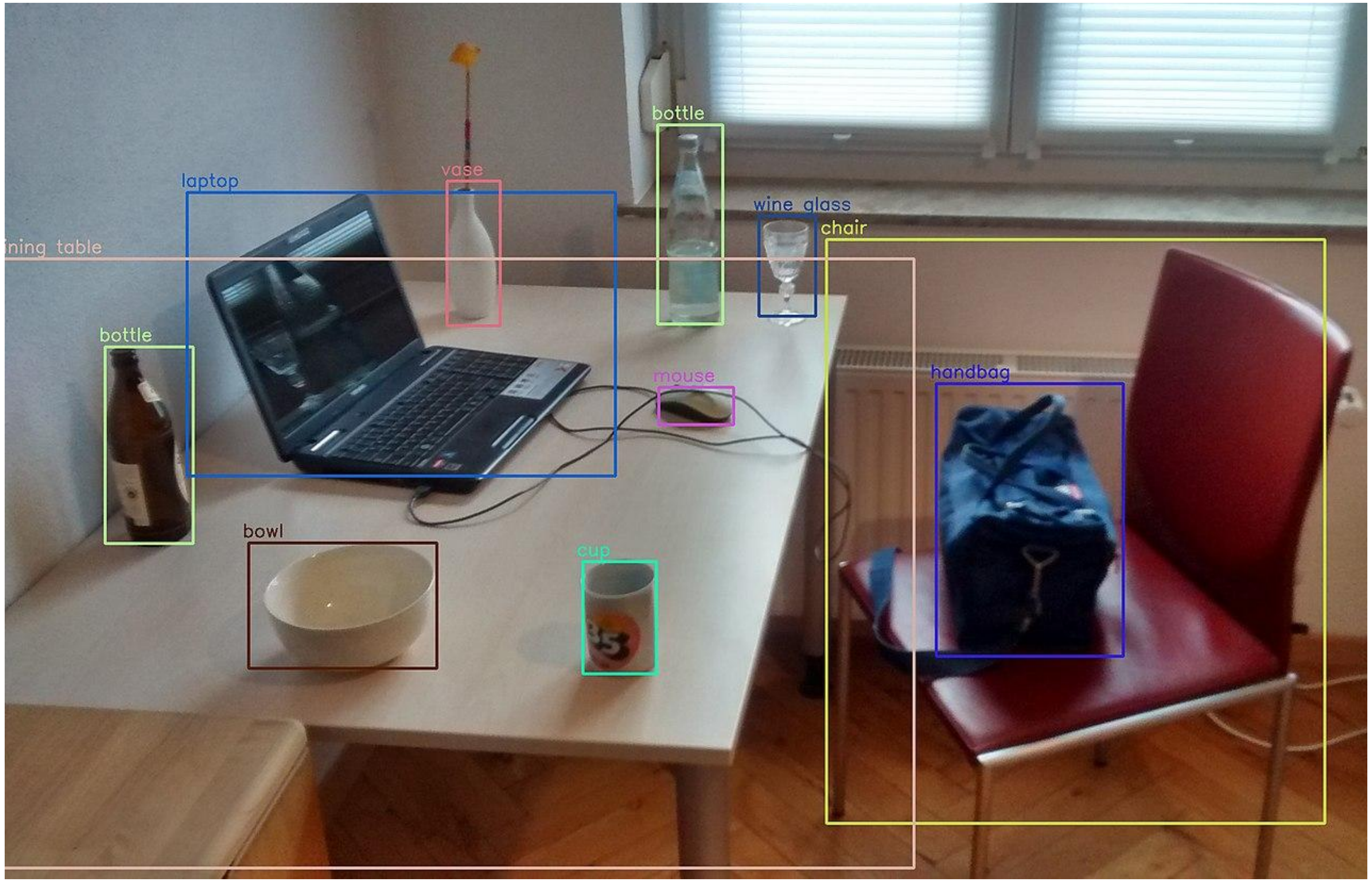
Object Detection

CMPUT 328

Nilanjan Ray

What is object detection?

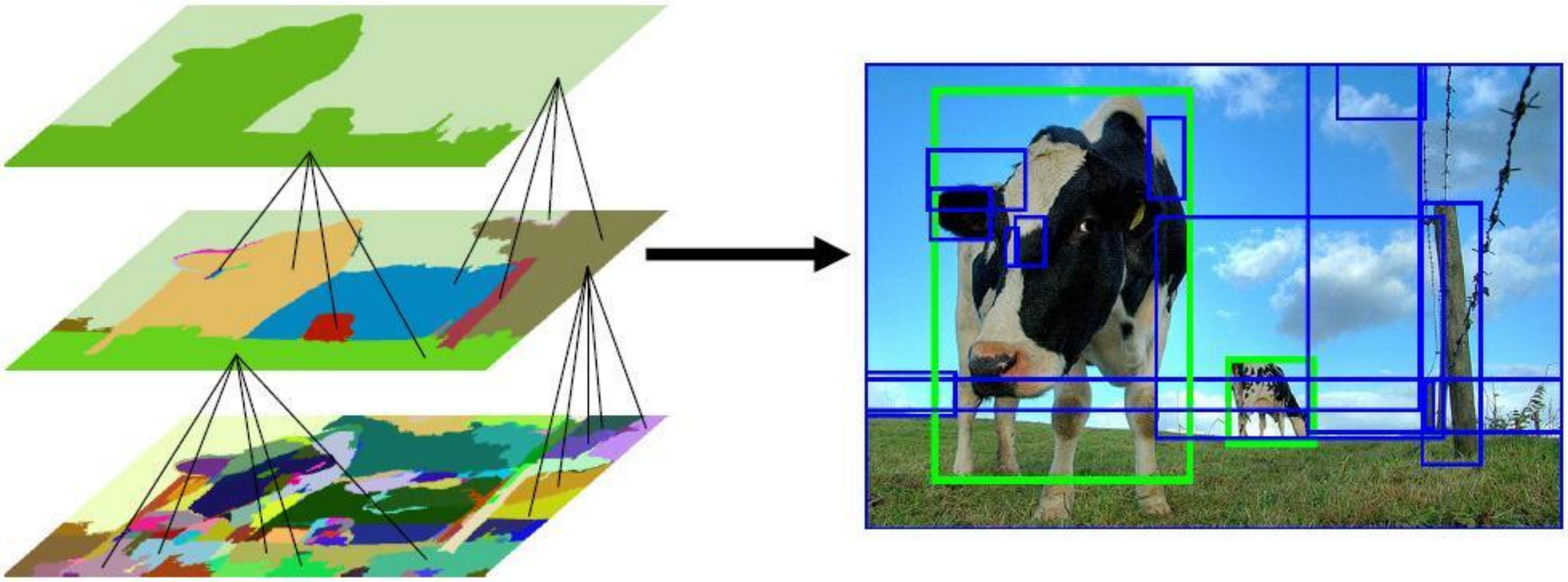


Picture source: https://en.wikipedia.org/wiki/Object_detection

One uniqueness about object detection

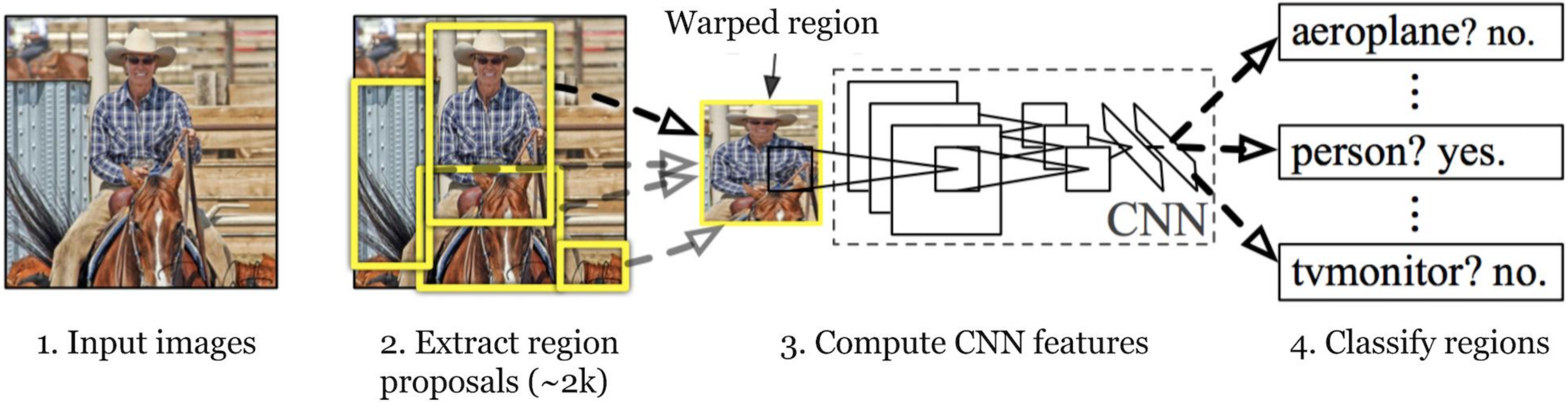
- For a test image the architecture does not know how many bounding boxes it must output.
- So, the length of output is a variable number.
- Over time there are several workarounds and methods came out to tackle this issue.
  - Older generation of object detectors: sliding window, region proposal using selective search
  - Not so older generation of detectors: region proposals using neural net, ROI pooling, anchor boxes, apply a threshold on “objectness”, merge nearby bounding boxes.
  - Latest generation of detectors: variable length sequential outputs using transformer architecture.

Region proposals: Selective search



Picture source: https://ivi.fnwi.uva.nl/isis/publications/bibtexbrowser.php?key=UijlingsIJCV2013&bib=all.bib

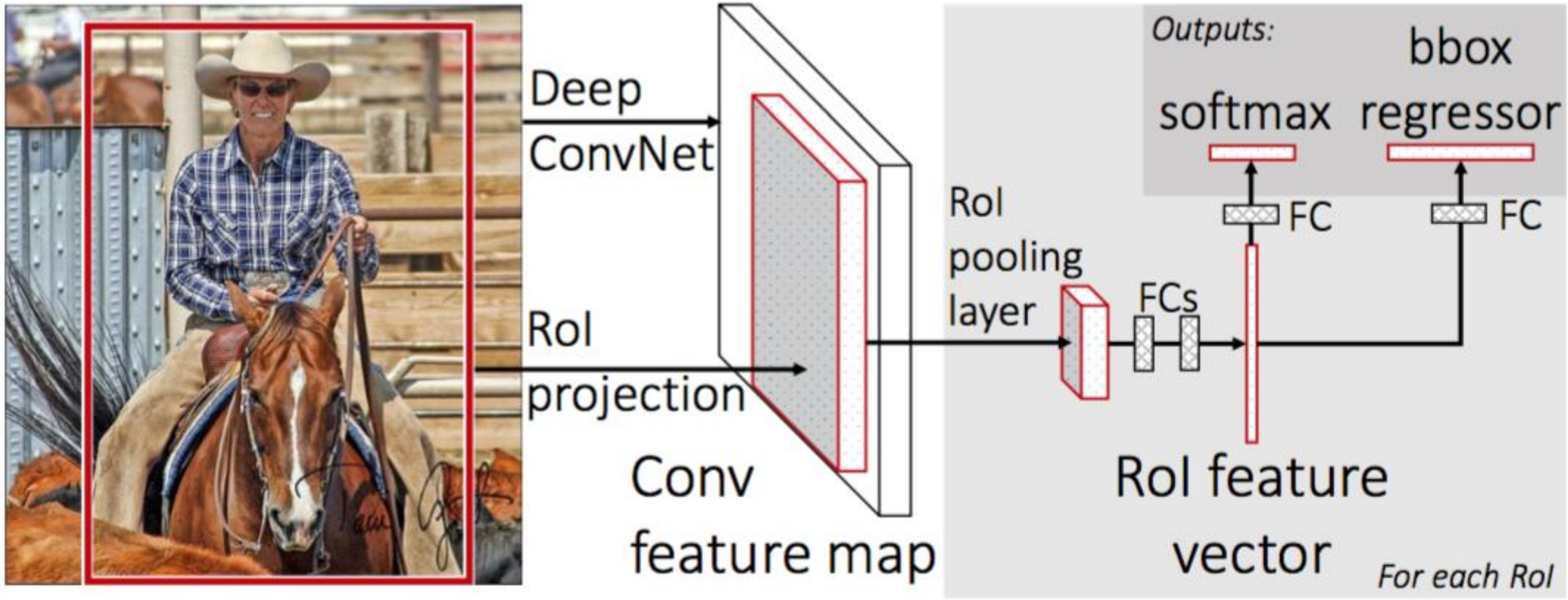
R-CNN



Picture source: https://arxiv.org/abs/1311.2524

Slow because we need to send ~2k cropped images through the CNN

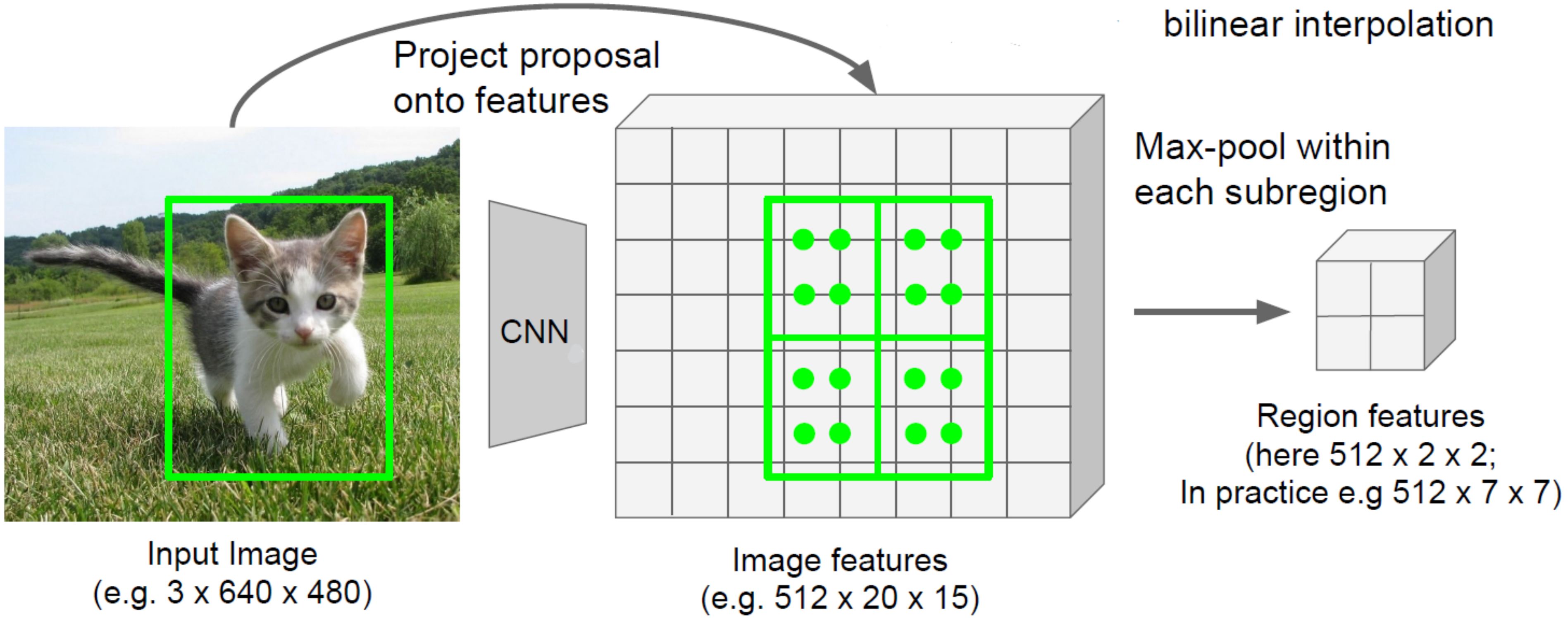
Fast R-CNN



Pass image only once through the CNN; Pool ROI features from the feature map for bounding box regression and classification; Fast because ~2k small feature (because of ROI pooling) maps now passes through a fully connected net.

Picture source: https://arxiv.org/pdf/1504.08083.pdf

ROI pooling



Picture source: http://cs231n.stanford.edu

Bounding box regressor

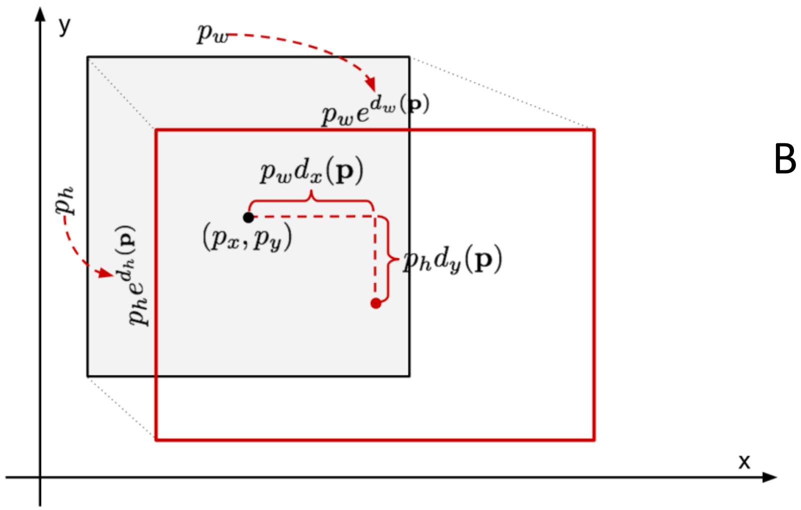
Bounding box proposal (4 numbers): $\mathbf{p} = (p_x, p_y, p_w, p_h)$ is transformed into:

$$\begin{cases} \hat{g}_x = p_w d_x(\mathbf{p}) + p_x \\ \hat{g}_y = p_h d_y(\mathbf{p}) + p_y \\ \hat{g}_w = p_w \exp(d_w(\mathbf{p})) \\ \hat{g}_h = p_h \exp(d_h(\mathbf{p})) \end{cases}$$

$d_i(\mathbf{p})$ represents a fully connected neural net having parameters **w** called bbox regressor

Ground truth bounding box (4 numbers): $\mathbf{g} = (g_x, g_y, g_w, g_h)$ is transformed into:

$$\begin{cases} t_x = (g_x - p_x)/p_w \\ t_y = (g_y - p_y)/p_h \\ t_w = \log(g_w/p_w) \\ t_h = \log(g_h/p_h) \end{cases}$$



Bbox Regression Loss function:

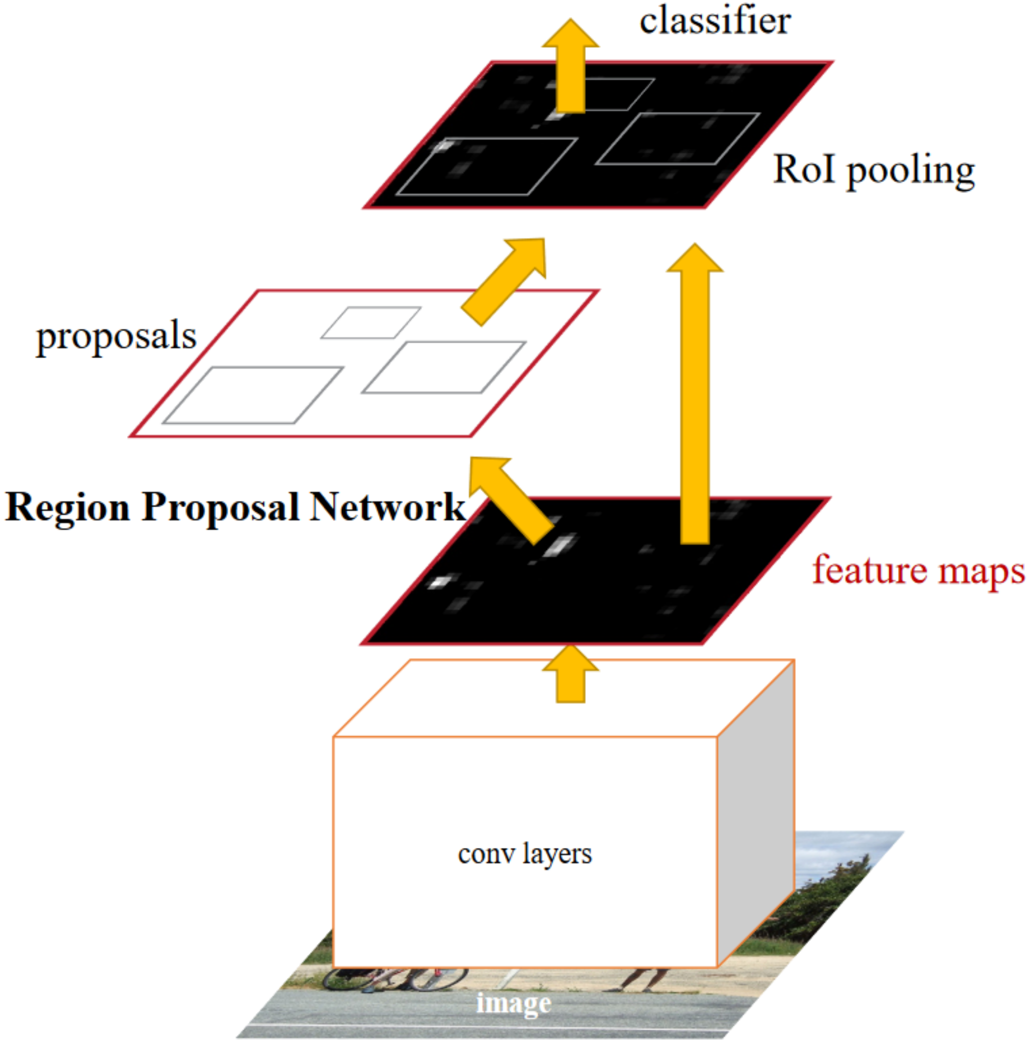
$$\mathcal{L}_{\text{reg}} = \sum_{i \in \{x,y,w,h\}} (t_i - d_i(\mathbf{p}))^2 + \lambda \|\mathbf{w}\|^2$$

https://arxiv.org/pdf/1311.2524.pdf

Faster R-CNN

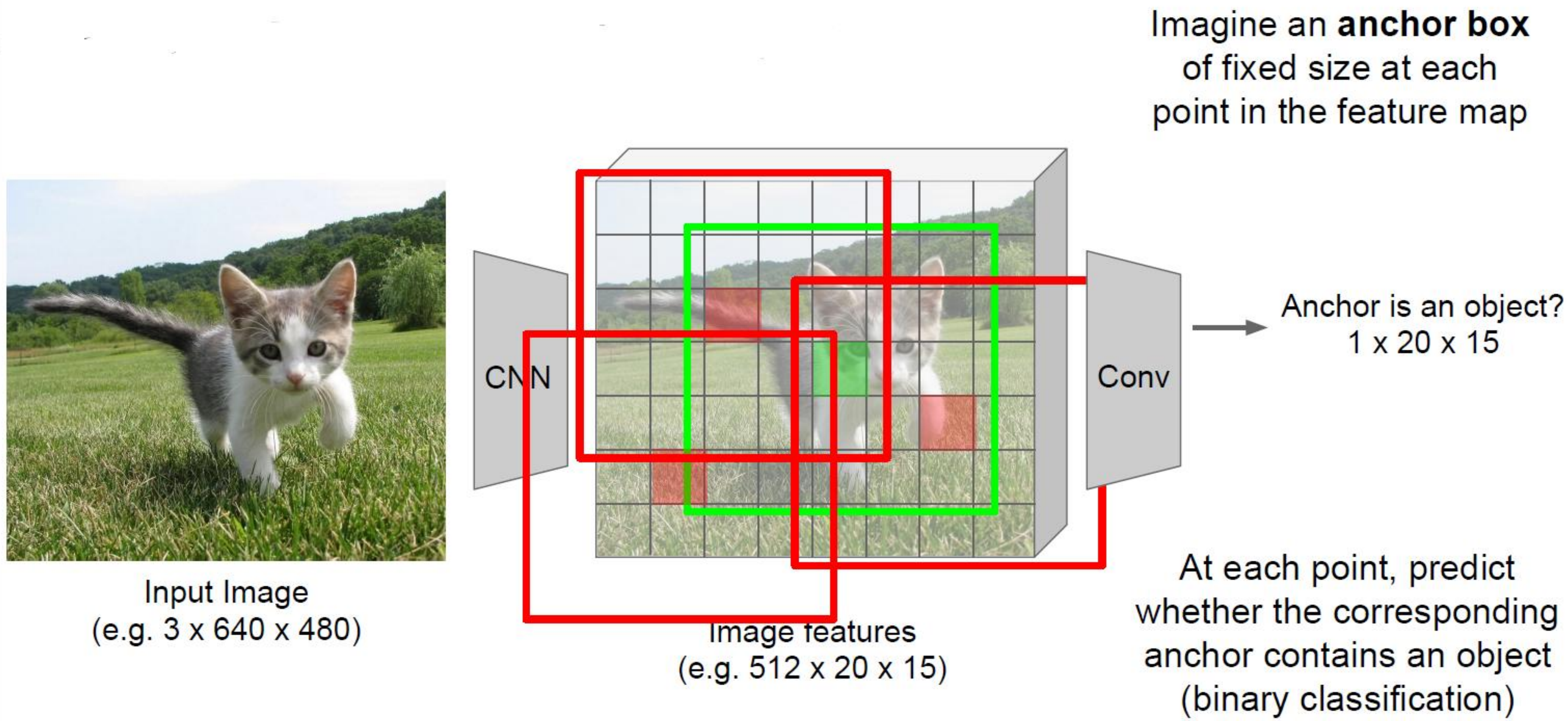
- Generate region proposals by a CNN (now we need anchors)
- Do ROI pooling as before
- Train in two stages

When we used selective search for region proposals we did not need anchors. Why?



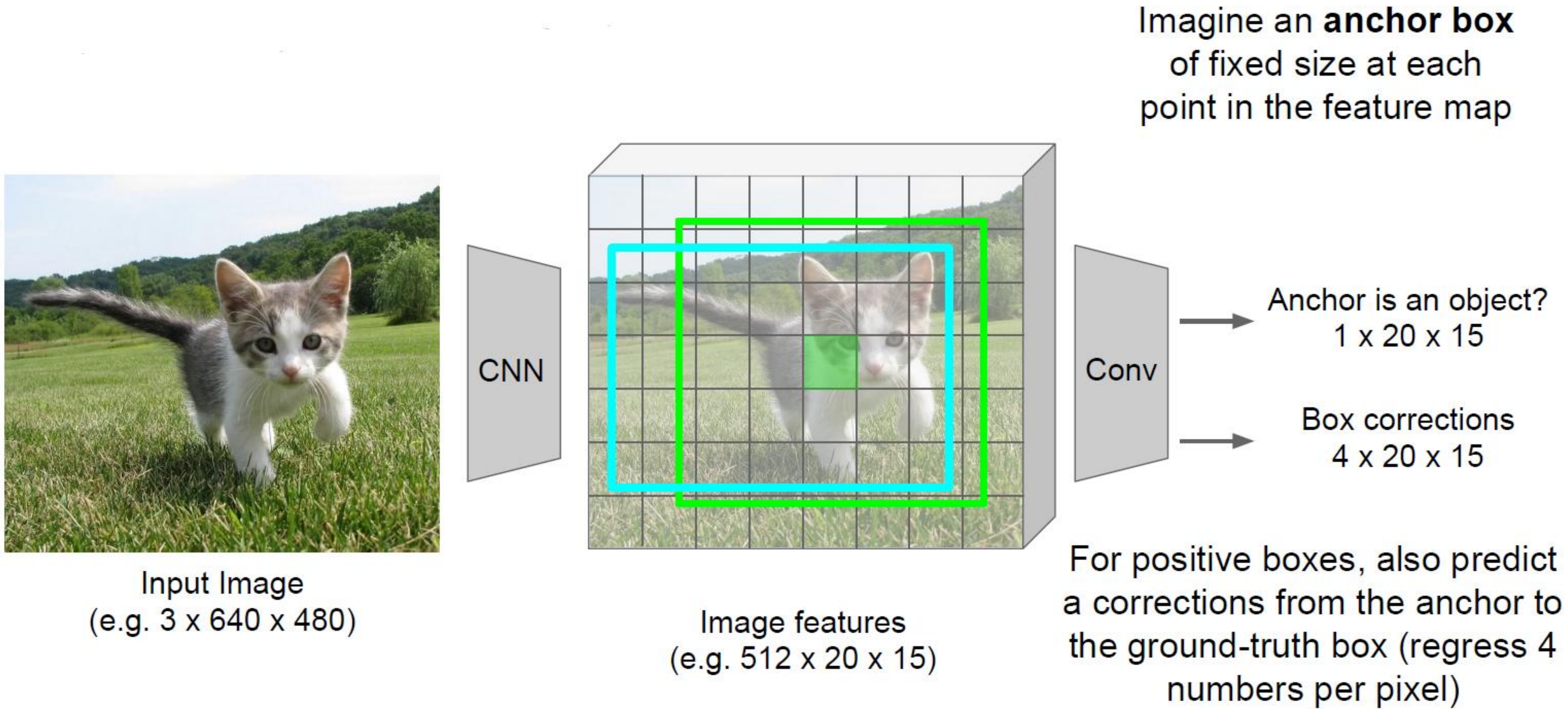
Picture source: https://arxiv.org/pdf/1506.01497.pdf

Region proposal network



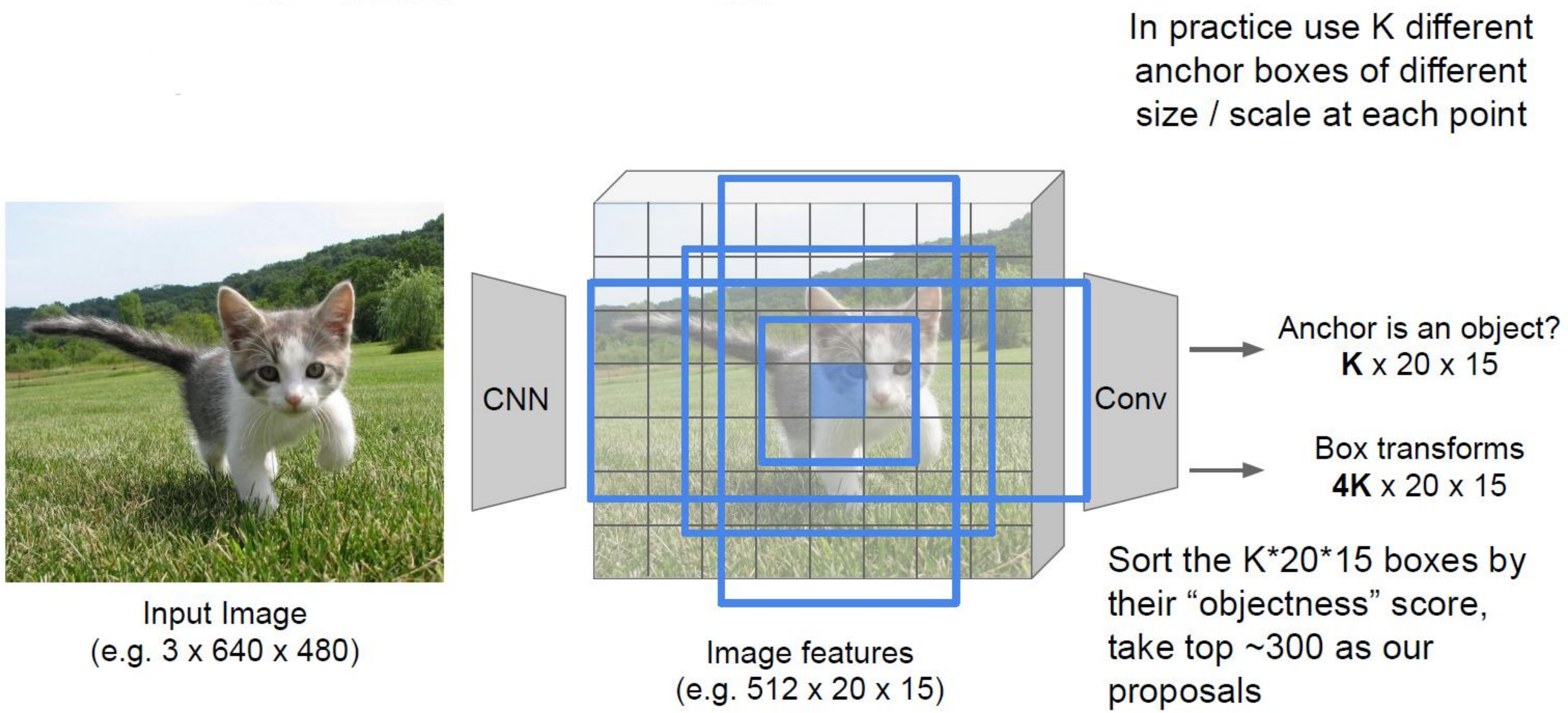
Picture source: http://cs231n.stanford.edu

Region proposal network…



Picture source: http://cs231n.stanford.edu

Region proposal network…



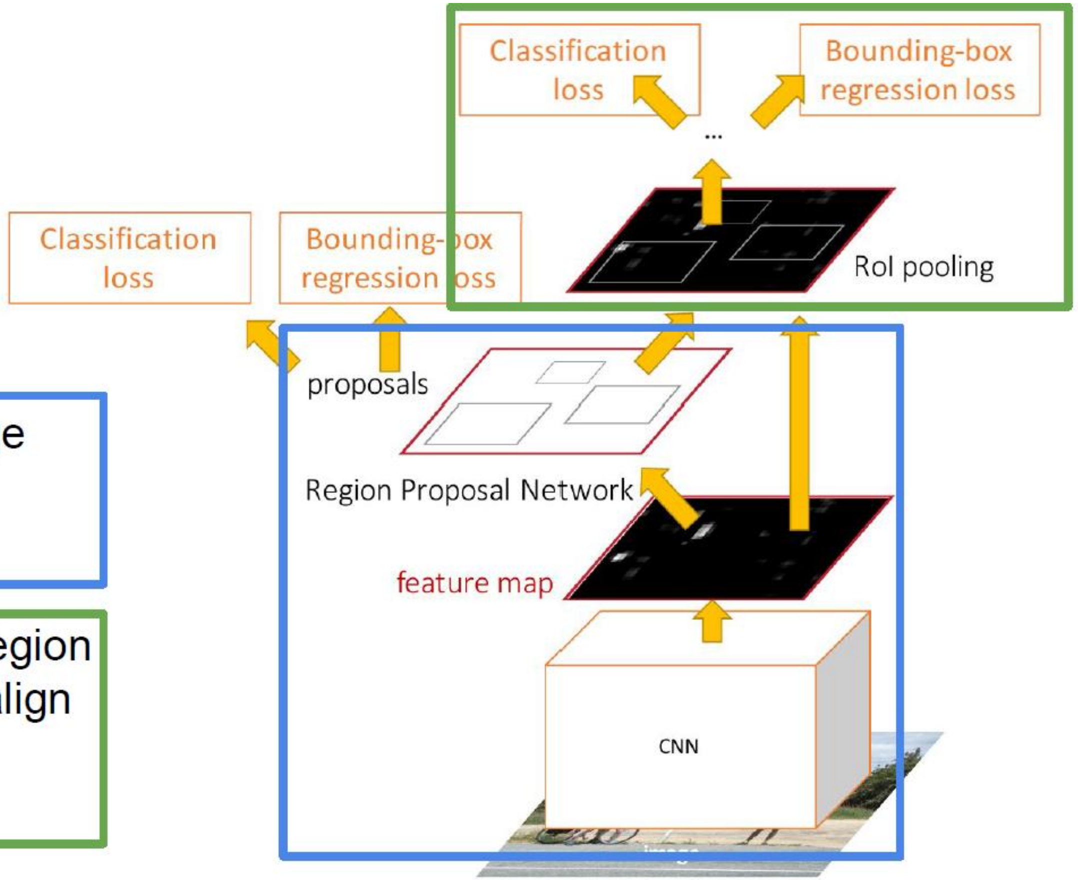
Picture source: http://cs231n.stanford.edu

Two-stage training in faster R-CNN

Faster R-CNN is a
Two-stage object detector

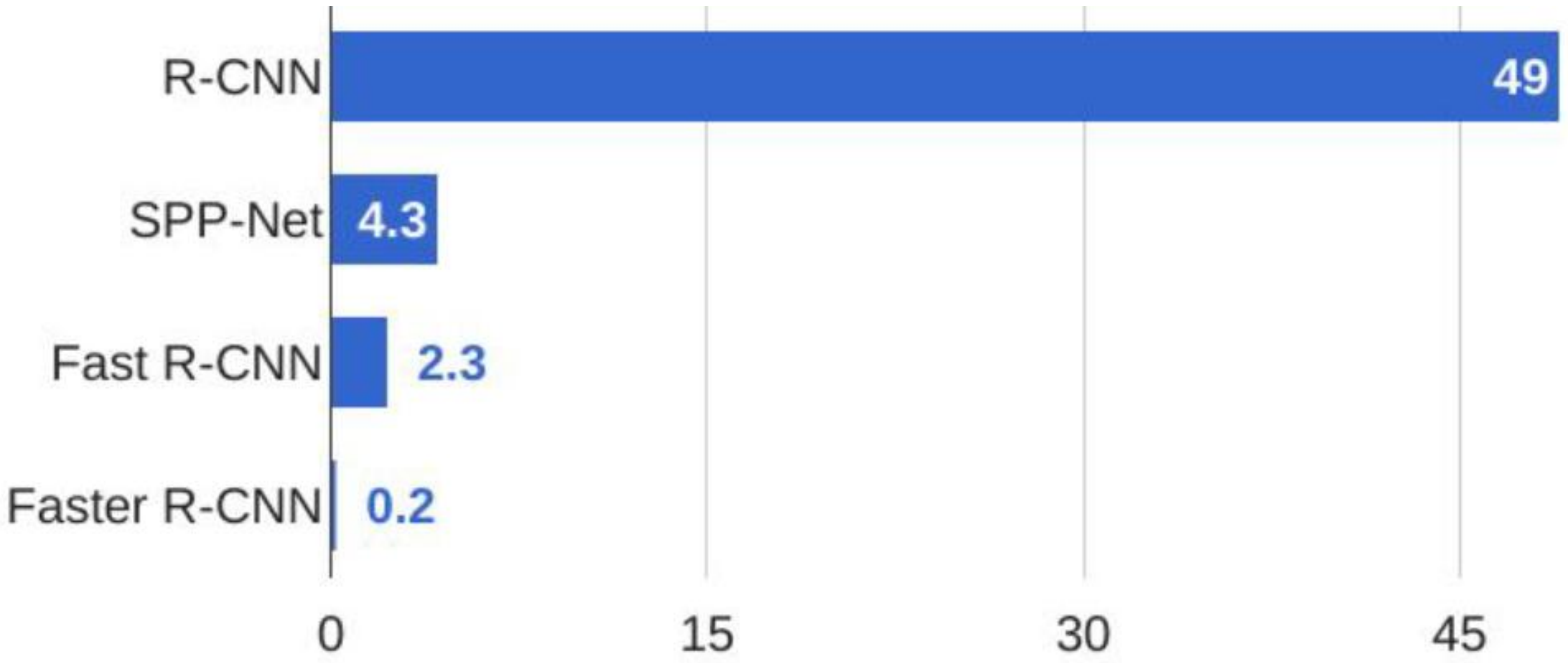
First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Picture source: http://cs231n.stanford.edu

Test time speed up (seconds per image)



YOLO: Single stage object detector

> 10x speed up over faster R-CNN

Apply threshold on confidence to select bounding boxes

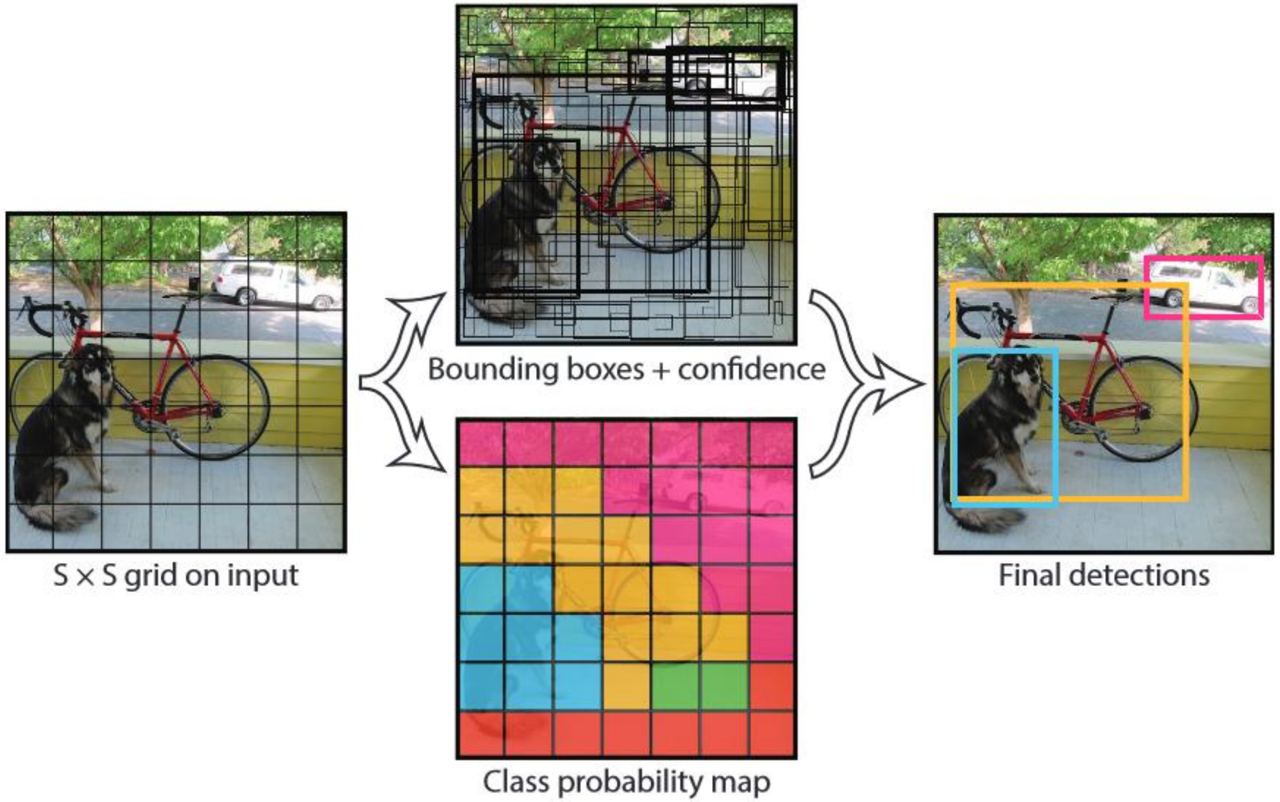
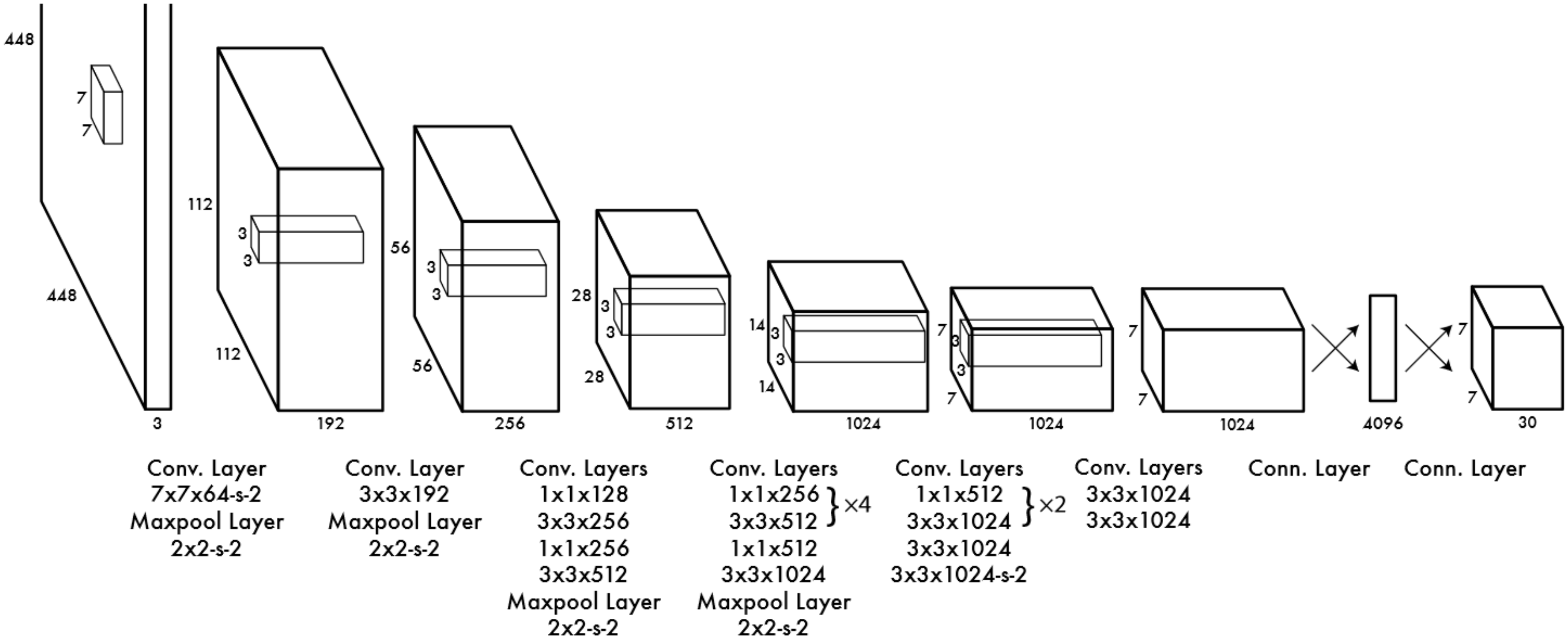


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Picture source: https://arxiv.org/pdf/1506.02640v5.pdf

YOLO architecture



Picture source: https://arxiv.org/pdf/1506.02640v5.pdf

Non-maximum suppression

The very last step in object detection

Selecting one bounding box out of so many nearby ones
Is it done during training too?

Source: https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/

RetinaNet: Another single stage object detector

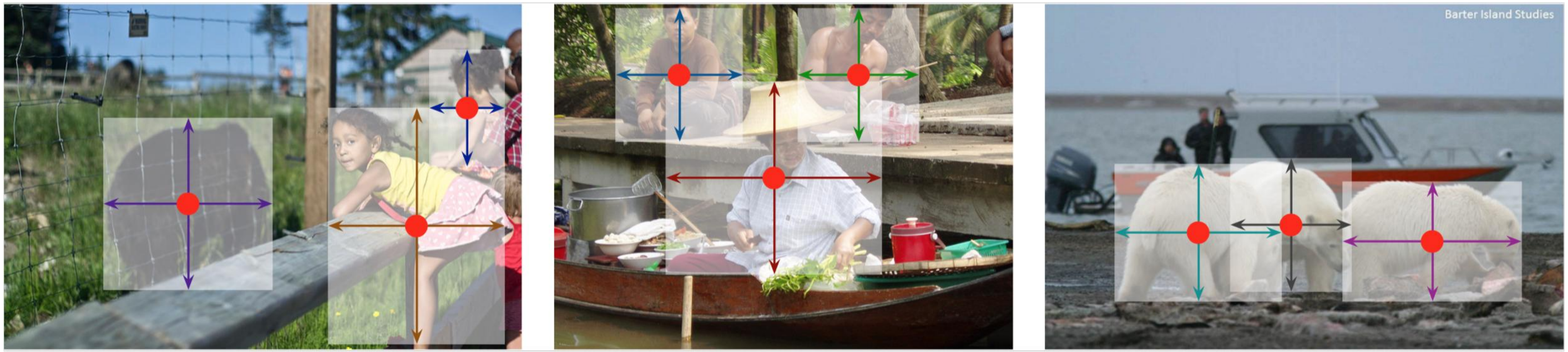
- An excellent blog: https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/

Anchorless object detection

- Two sub-categories
  - Predict heatmap and treat object as points (https://github.com/xingyizhou/CenterNet)
  - Predict a set, one element at a time sequentially as in transformer (https://arxiv.org/pdf/2005.12872.pdf)

CenterNet

- Objects as points



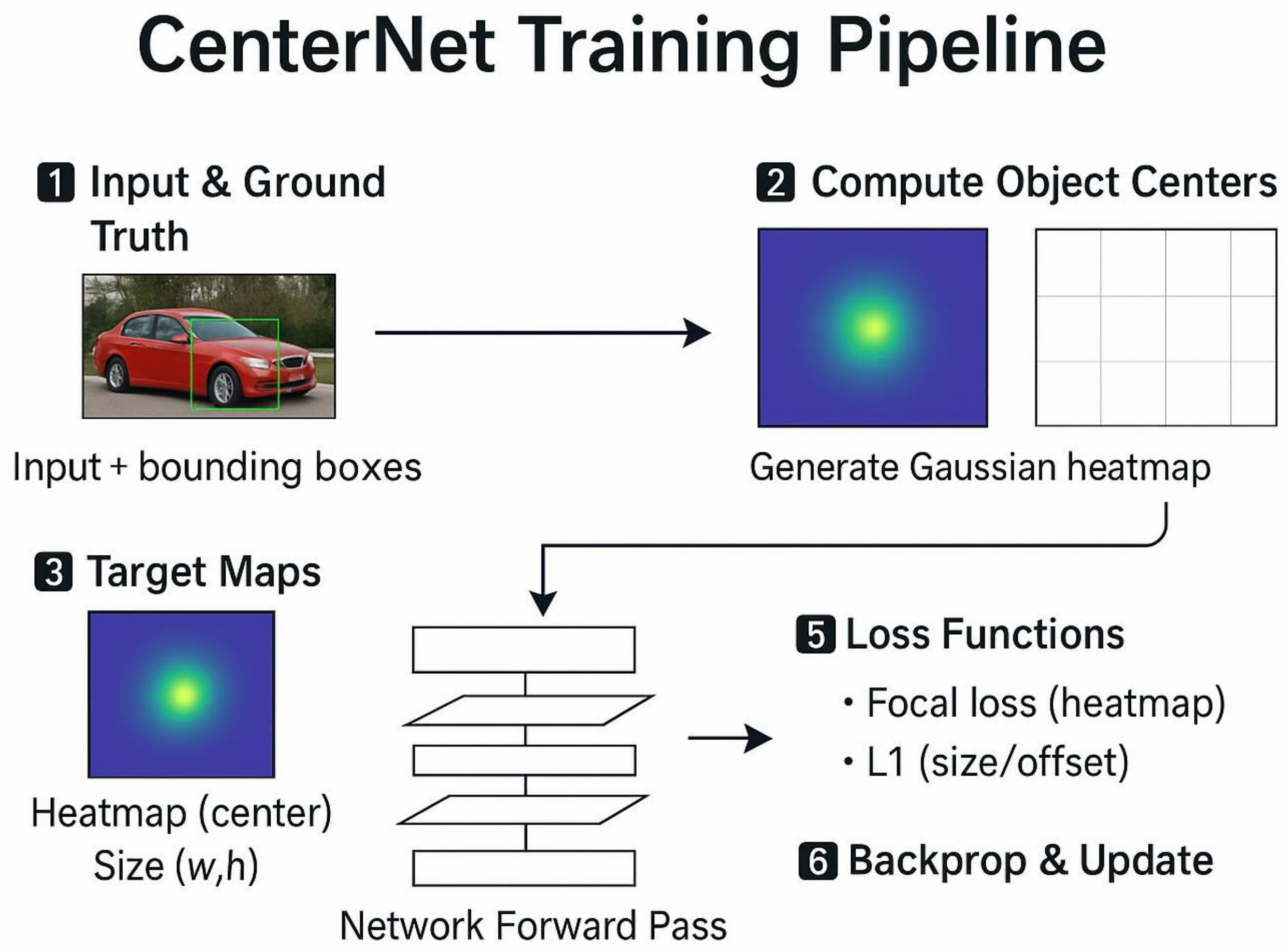
Picture source: https://github.com/xingyizhou/CenterNet?tab=readme-ov-file

CenterNet training

- **From each ground-truth object center →** generate a **Gaussian heatmap**, and **match it** with the **heatmap predicted by the network** (using focal loss)

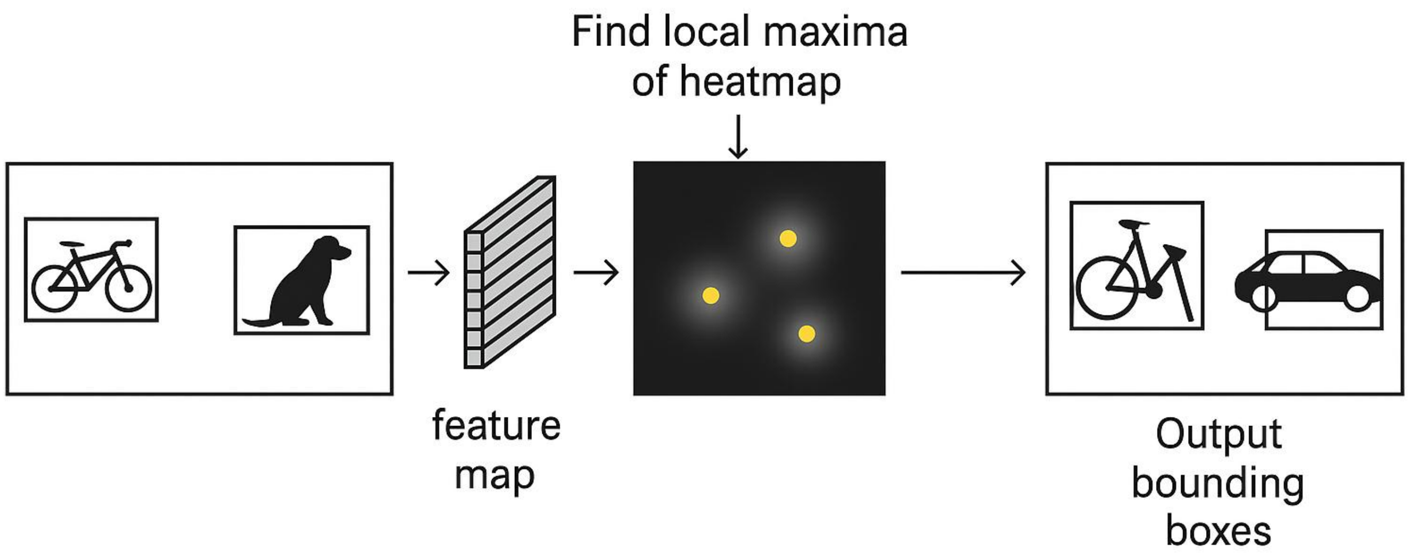
- **At each ground-truth center location**, feed the **feature map** to the network heads to predict:
  - **Bounding box size (w, h)**
  - **Sub-pixel offset (Δx, Δy)**

- **Regress** these predictions to the **ground-truth bounding box** (using L1 loss)



CenterNet deployment (aka testing)

- **Input image → CNN backbone → detection heads**
→ Network predicts **heatmap**, **size**, and **offset** maps.
- **Find local maxima** in the predicted heatmap
→ Each peak corresponds to a detected object center.
- **For each detected center:**
- Read predicted width, height, and offset values.
- Reconstruct the bounding box in image space.
- **Apply confidence threshold** (and optional NMS).
→ Output a *variable number* of final bounding boxes.



End-to-End Object Detection with Transformers (DETR)

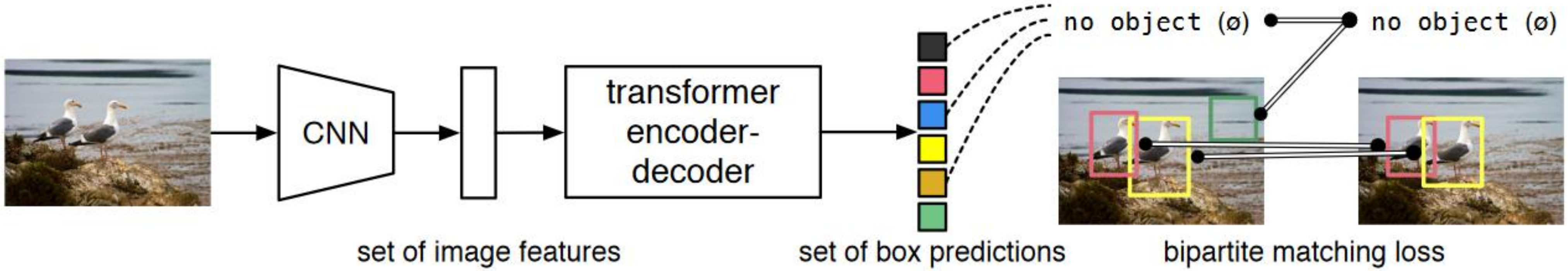


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "no object" (∅) class prediction.

https://arxiv.org/pdf/2005.12872.pdf

Hungarian Matching (HM) in DETR

- **HM step is non-differentiable**
→ It contains discrete **argmin/argmax** operations, which are not differentiable.
- **Training still works**
→ HM is performed **outside the computation graph** (e.g., in a torch.no_grad() block).
→ It provides **matching indices** between predicted and ground-truth boxes.
- **Loss computation**
→ The **loss function** uses these indices to compute classification and regression losses.
→ These losses are **fully differentiable** with respect to network outputs.
- **Gradient flow**
→ No gradient flows through the HM step itself.
→ Effectively, the gradient is passed **as if via a straight-through estimator (STE)** —
i.e., the assignment is treated as fixed during backpropagation.

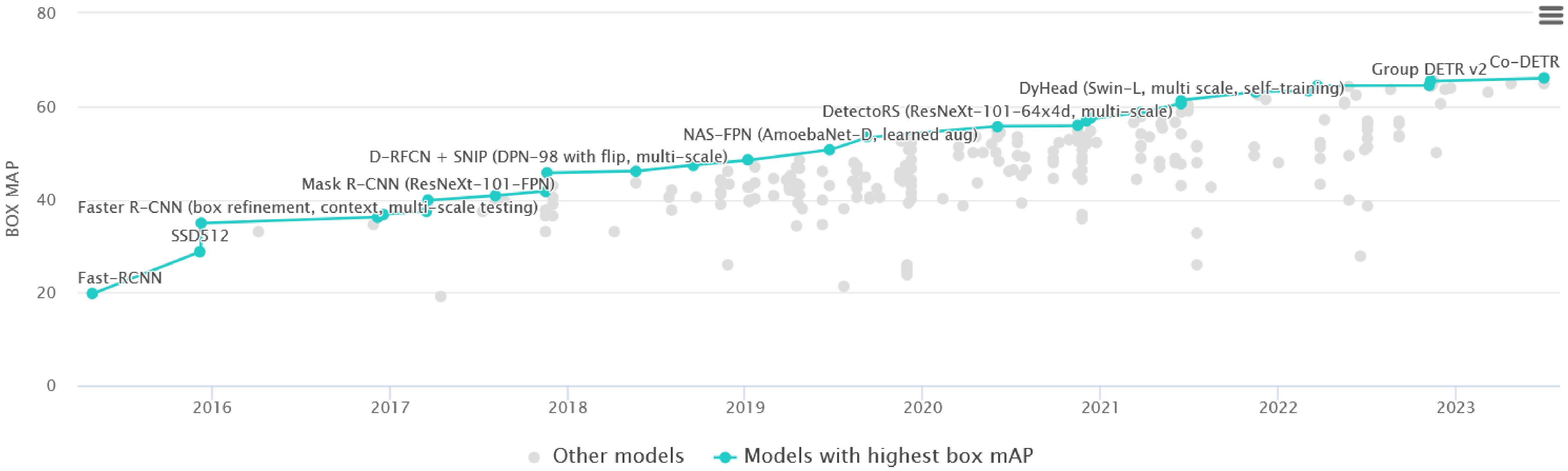
HM non-differentiability: What did we miss?

•Gradients do **not** flow through the assignment step →
•model cannot learn how small changes in predictions would change matching.
•This causes:
•**Noisy early supervision** (random matches).
•**Discontinuous loss surface** when assignments flip.
•**Slow convergence** (hundreds of epochs on COCO).

How later works addressed HM non-differentiability

- **Soft or Differentiable Matching**
  - *Soft-DETR, Gumbel-Sinkhorn, Optimal Transport relaxations* → replace hard Hungarian with **soft assignment matrices** so gradients flow through matching.
- **Better Initialization & Query Guidance**
  - *Conditional-DETR, Anchor-DETR* → condition queries on spatial priors to reduce matching ambiguity.
- **Stabilized Early Training**
  - *DN-DETR (Denoising DETR)* → add noised ground-truth queries for easier alignment and faster convergence.
- **Improved Feature Sampling**
  - *Deformable DETR* → multi-scale deformable attention; improves gradient flow and convergence speed even with non-diff matching.

Performance comparisons on COCO



https://paperswithcode.com/sota/object-detection-on-coco

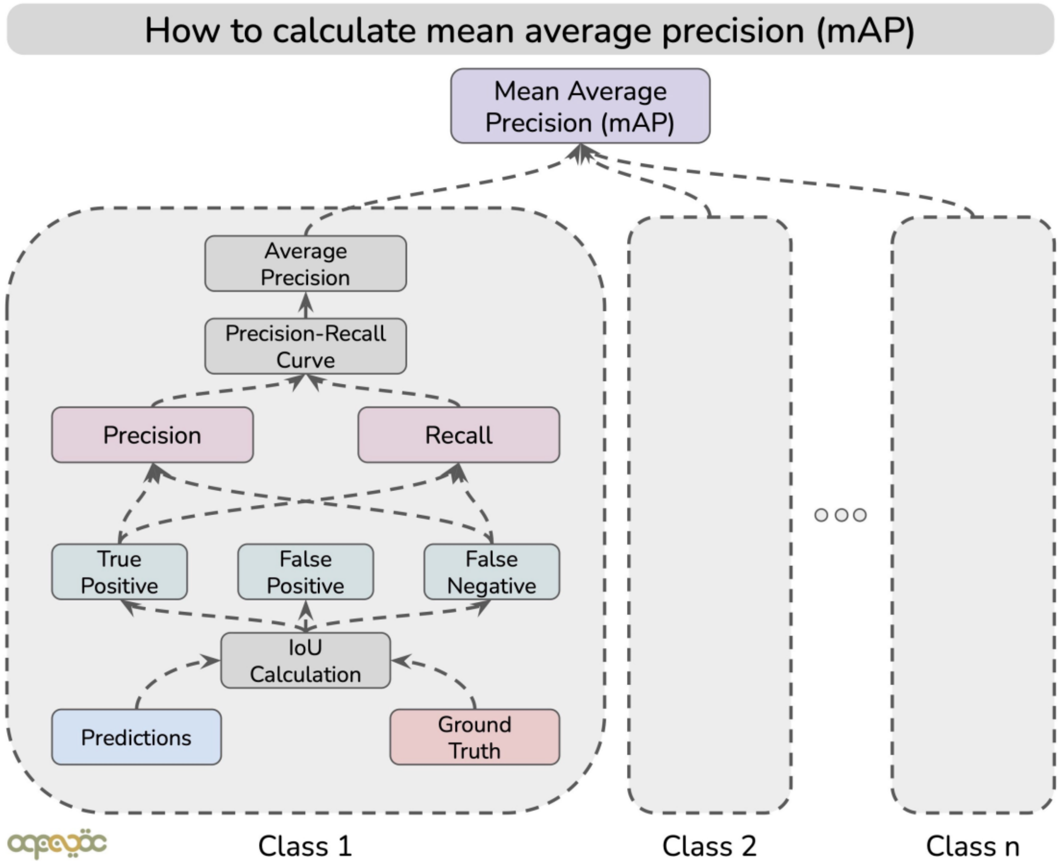
Tutorials

- https://pseudo-lab.github.io/Tutorial-Book-en/chapters/en/object-detection/intro.html

- https://detectron2.readthedocs.io/en/latest/tutorials/index.html

Evaluating object detection

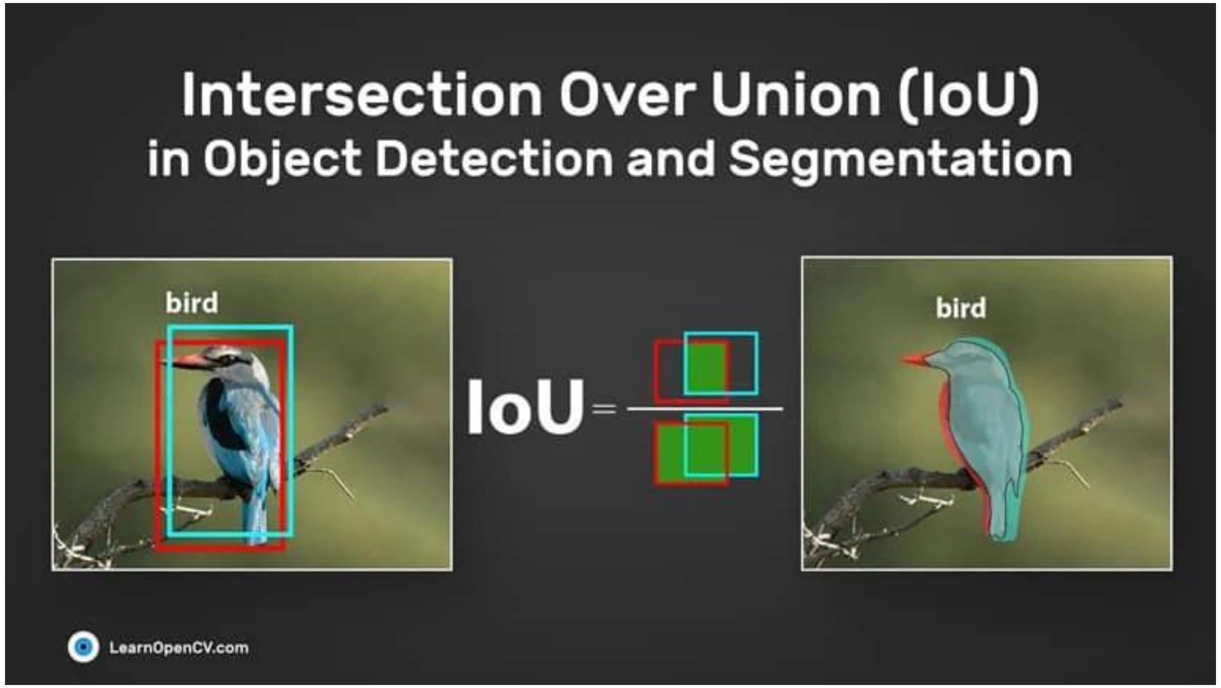
mAP



https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

Intersection over union

- IOU

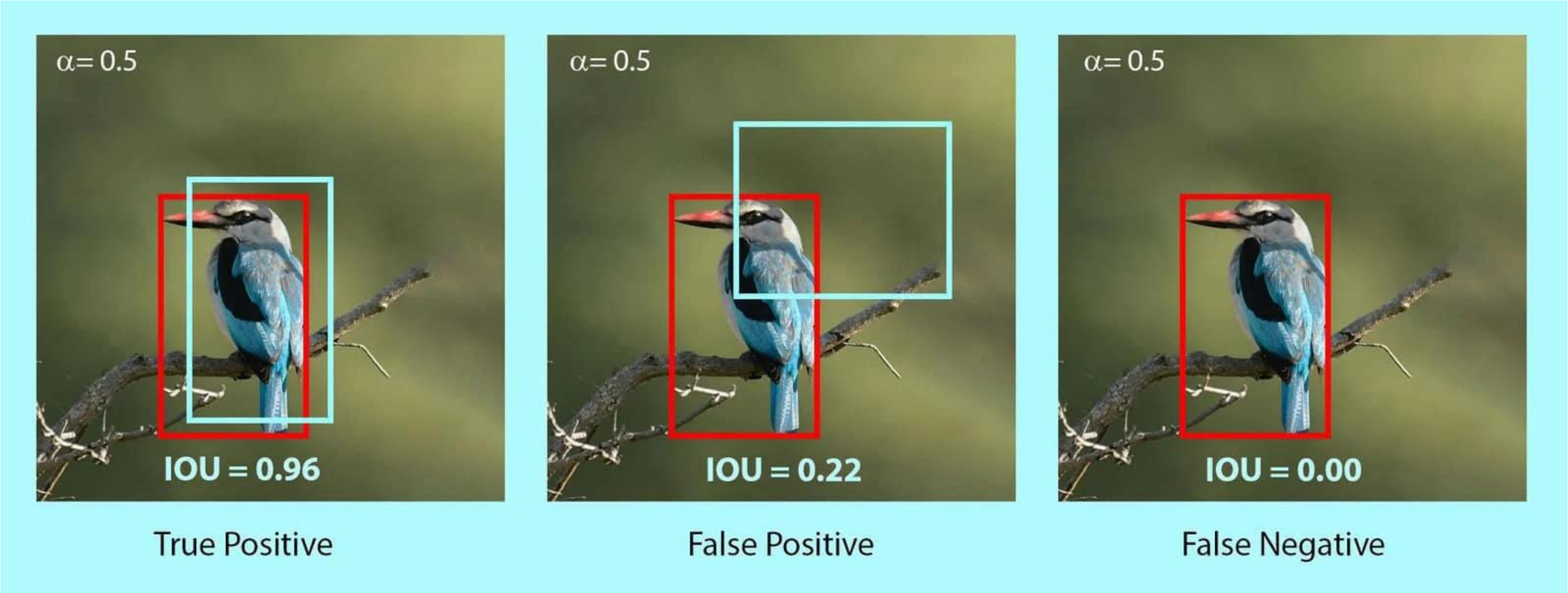


Picture source: https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/

TP, FP, FN, TN

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP),** **hit**	**False negative (FN),** **type II error, miss,** **underestimation**
	Negative (N)	**False positive (FP),** **type I error, false alarm,** **overestimation**	**True negative (TN),** **correct rejection**

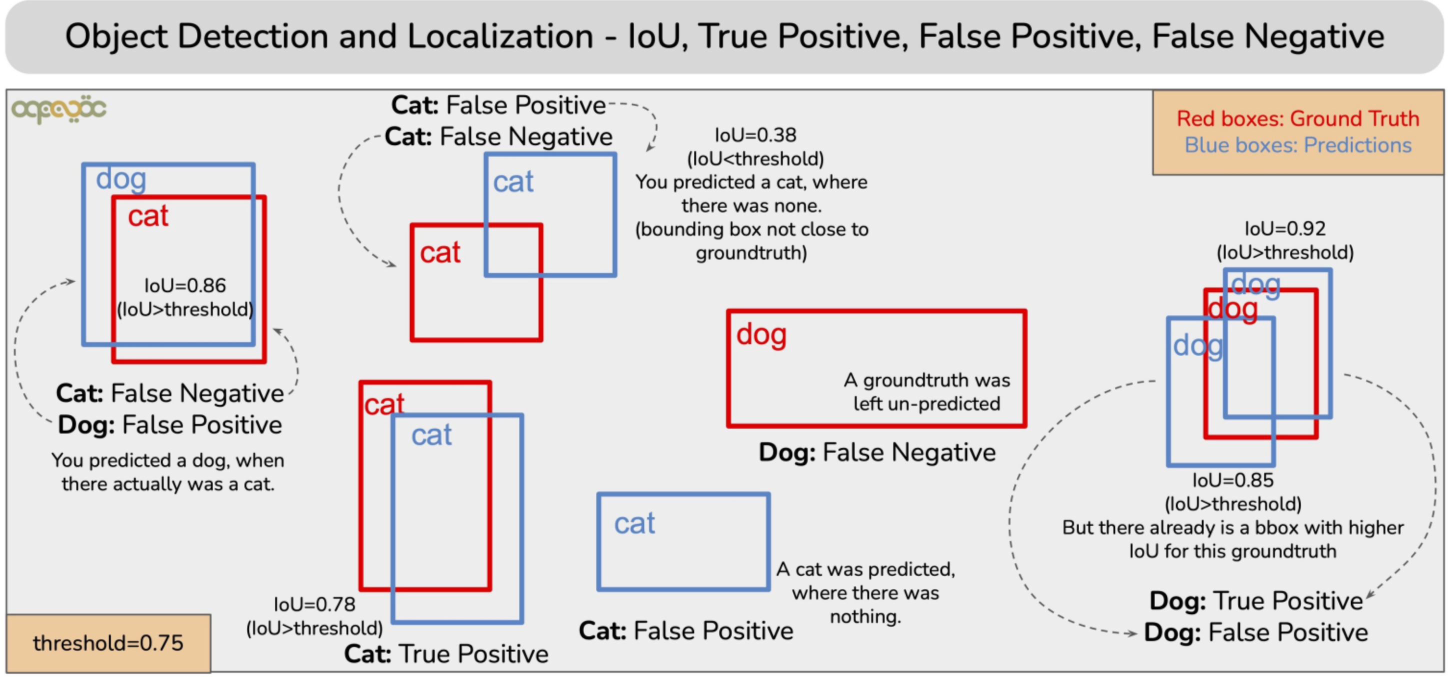
TP, FP, FN in object detection

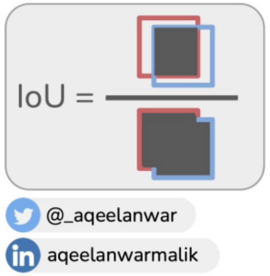


α is the IOU threshold

Picture source: https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/

Object detection - examples





Threshold	Class	# GroundTruth	# predictions	TP	FP	FN	Precision	Recall
0.75	Cat	3	3	1	2	2	1/3	1/3
	Dog	2	3	1	2	1	1/3	1/2
0.35	Cat	3	3	2	1	1	2/3	2/3
	Dog	2	3	1	2	1	1/3	1/2

https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

Precision and recall in object detection

Precision and Recall in Machine Learning

For each class

$$Precision = \frac{Correct\ Predictions}{Total\ Predictions} = \frac{TP}{TP + FP}$$

$$Recall = \frac{Correct\ Predictions}{Total\ GroundTruth} = \frac{TP}{TP + FN}$$

Class	# GroundTruth	# predictions	TP	FP	FN	Precision	Recall
Cat	10	5	4	1	6	4/5 (80%)	4/10 (40%)
Dog	10	10	8	2	2	8/10 (80%)	8/10 (80%)

The classifier is ***precise*** in what it predicts. When it says it is a cat (dog), it is correct 80% of the time. However, if there is a cat (dog) in an image the classifier can only detect it 50% (80%) of the time. Hence the model has a hard time ***recalling*** cats.

https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

AP and mAP

For a single class, area under precision-recall curve:

$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$$

where P_n and R_n are precision and recall for the n^{th} threshold.

mAP is the mean of AP's over all classes

Maximum and ideal mAP for an algorithm is 1

What does mAP measure?

https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/