Neurocomputing

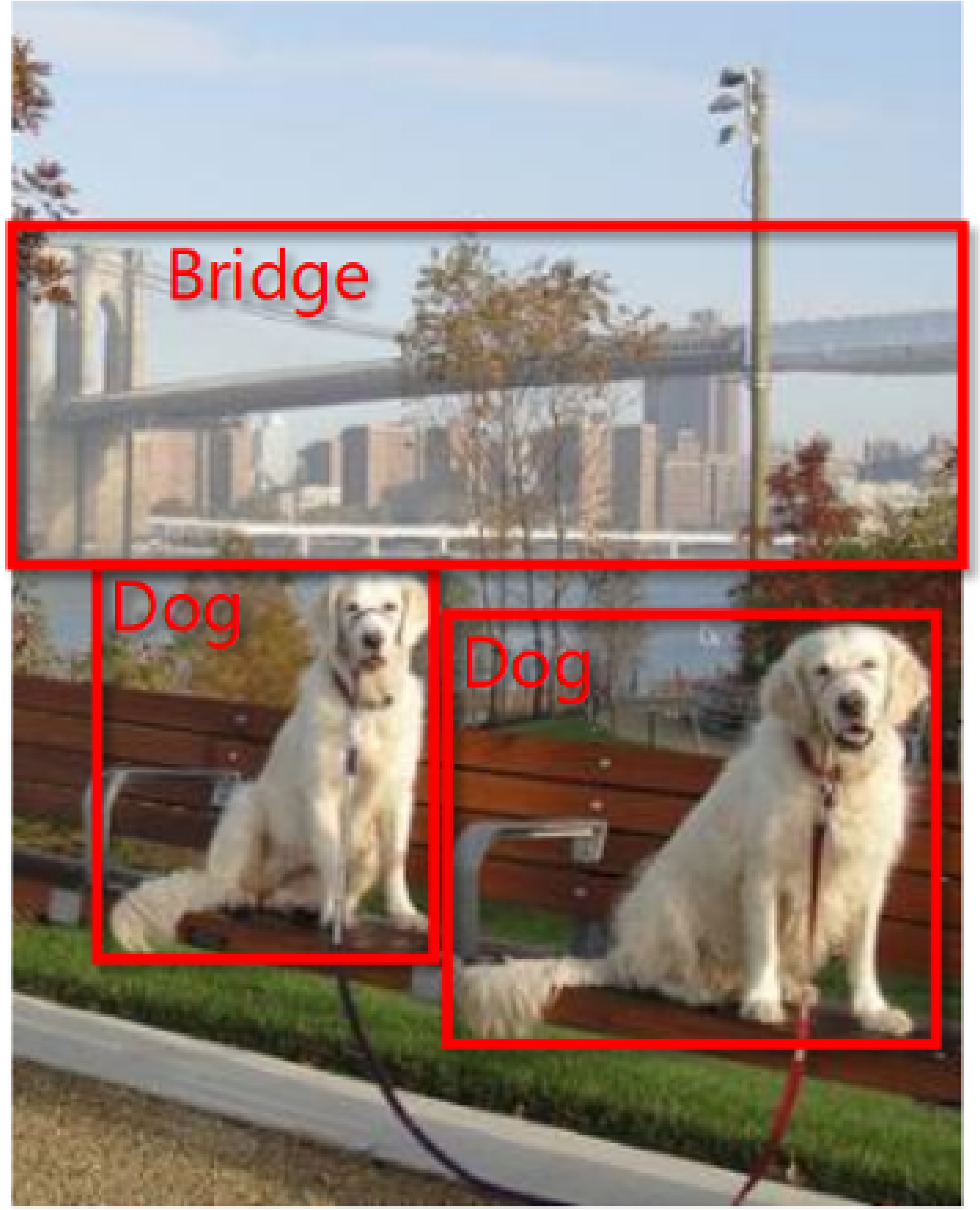
Object detection

Julien Vitay
Professur für Künstliche Intelligenz - Fakultät für Informatik

1 - Object detection

Object recognition vs. object detection

Classification, easy these days

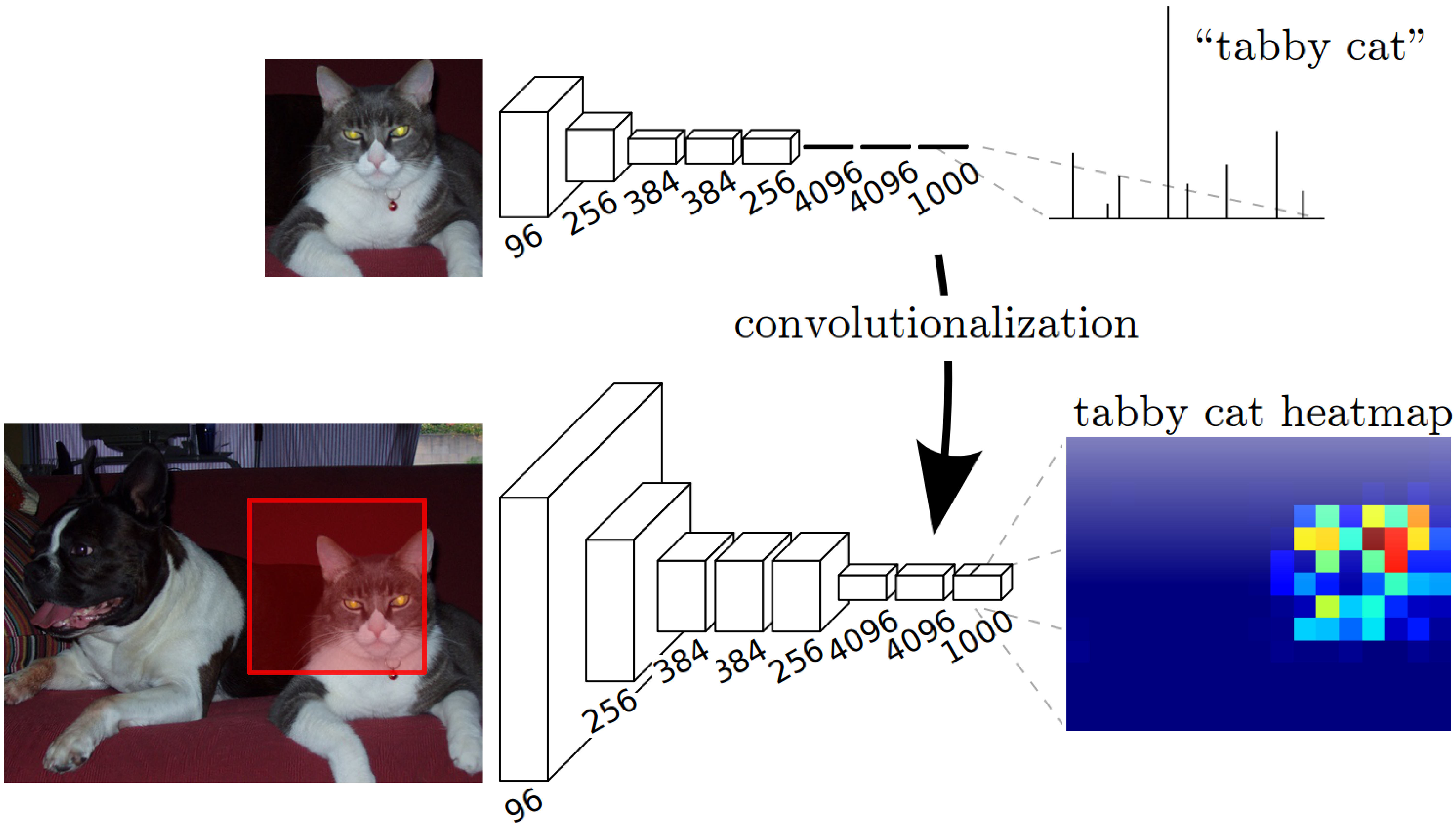


Object detection, still a lot harder

Source: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

Object detection with heatmaps

- A naive and very expensive method is to use a trained CNN as a high-level filter.
- The CNN is trained on small images and convolved on bigger images.
- The output is a heatmap of the probability that a particular object is present.

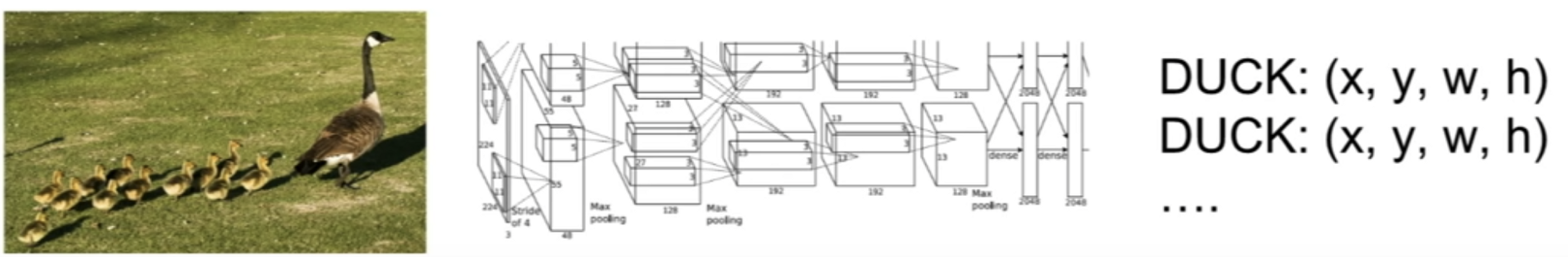


Source: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

PASCAL Visual Object Classes Challenge

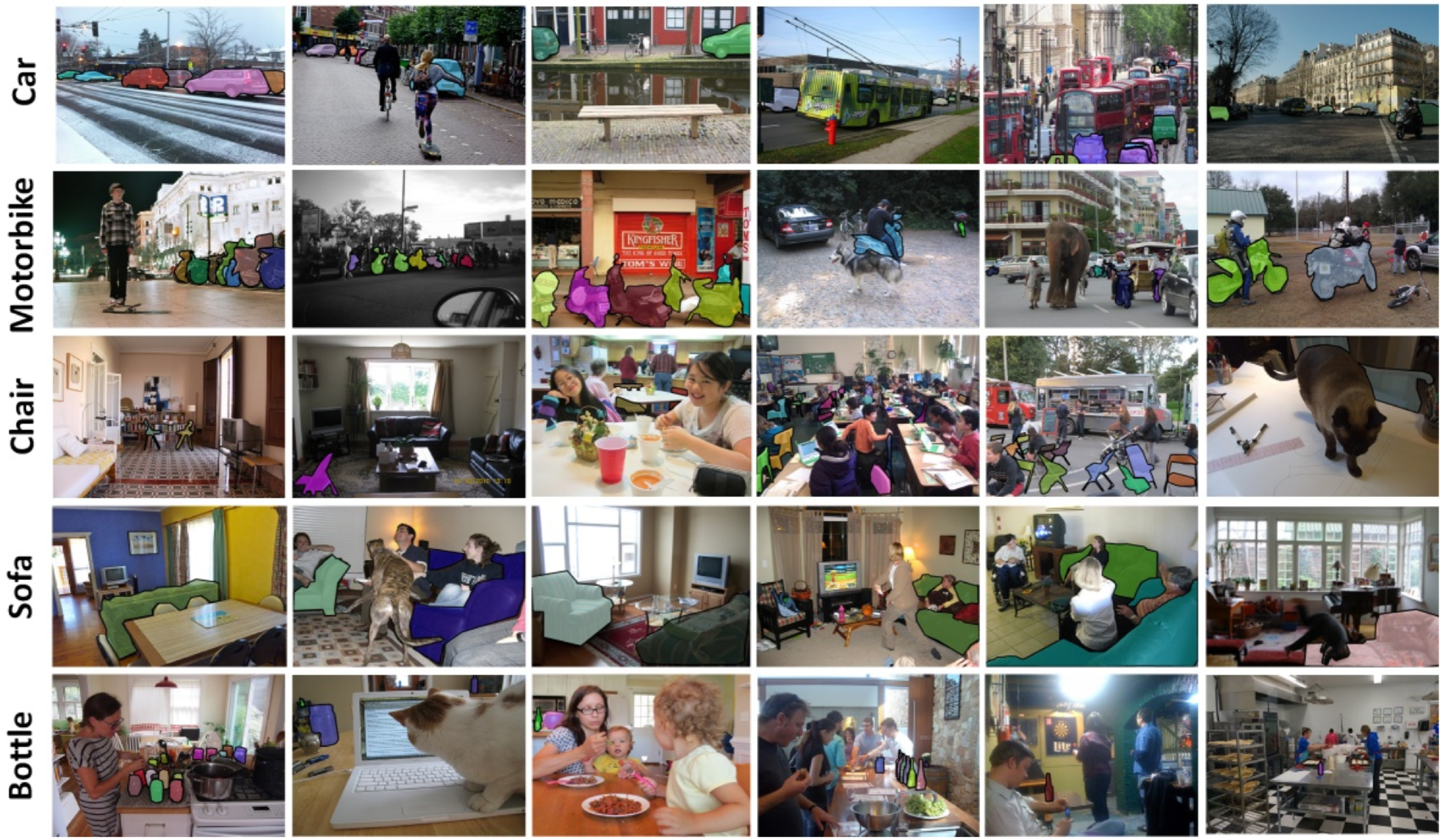
- The main dataset for object detection is the **PASCAL** Visual Object Classes Challenge:
  - 20 classes
  - ~10K images
  - ~25K annotated objects
- It is both a:
  - **Classification** problem, as one has to recognize an object.
  - **Regression** problem, as one has to predict the coordinates (x, y, w, h) of the bounding box.

Source: http://host.robots.ox.ac.uk/pascal/VOC/voc2008/



Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

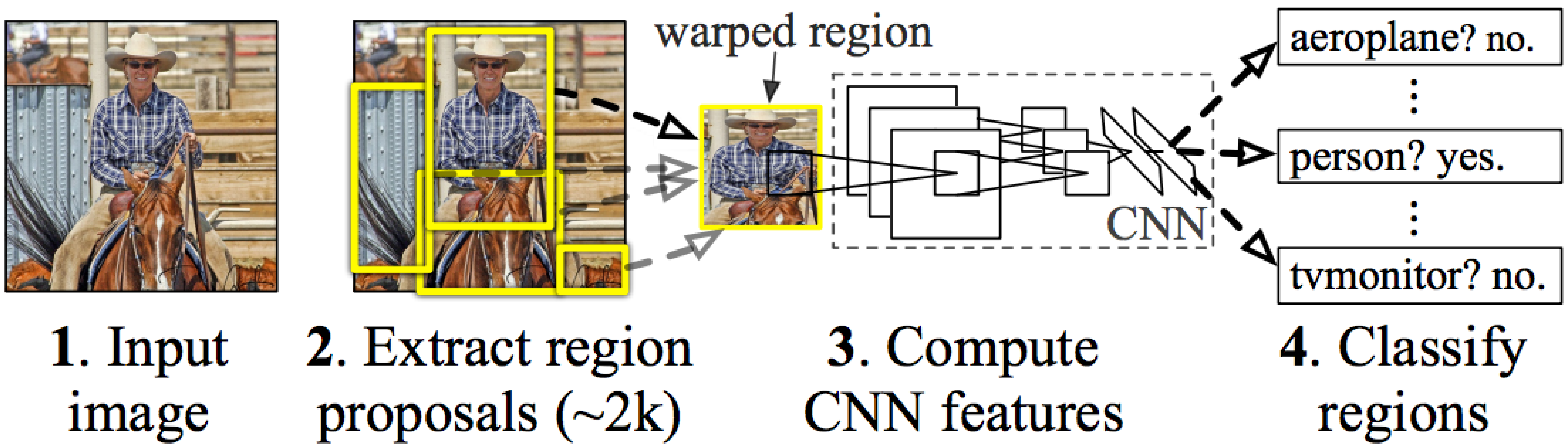
MS COCO dataset (Common Objects in COntext)



Source: http://cocodataset.org

- 330K images, 80 labels.
- Also contains data for semantic segmentation, caption generation, etc.

R-CNN : Regions with CNN features



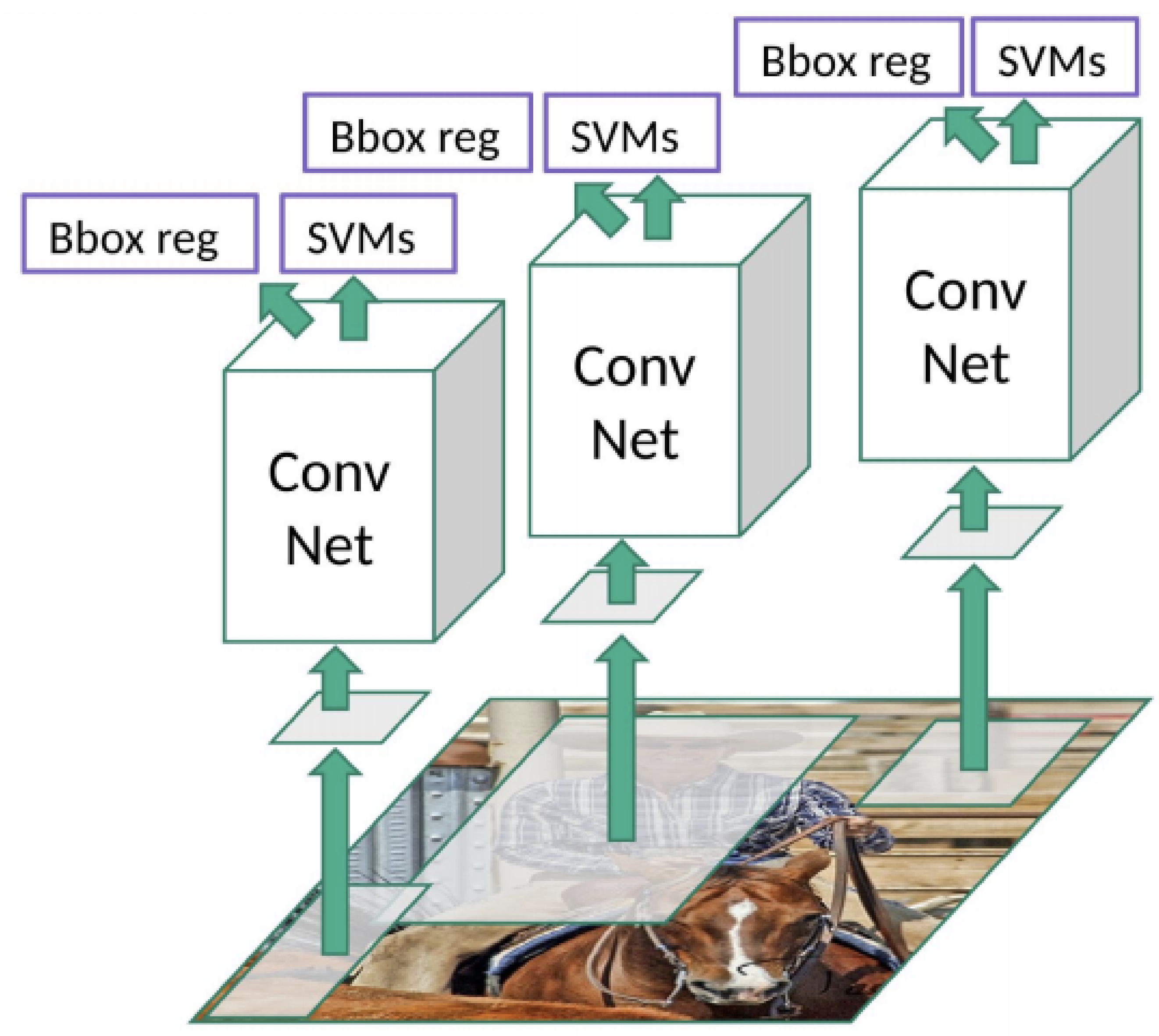
1. Bottom-up region proposals (selective search) by searching bounding boxes based on pixel info.
2. Feature extraction using a pre-trained CNN (AlexNet).
3. Classification using a SVM (object or not; if yes, which one?)
4. If an object is found, linear regression on the region proposal to generate tighter bounding box coordinates.

> Selective search: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf

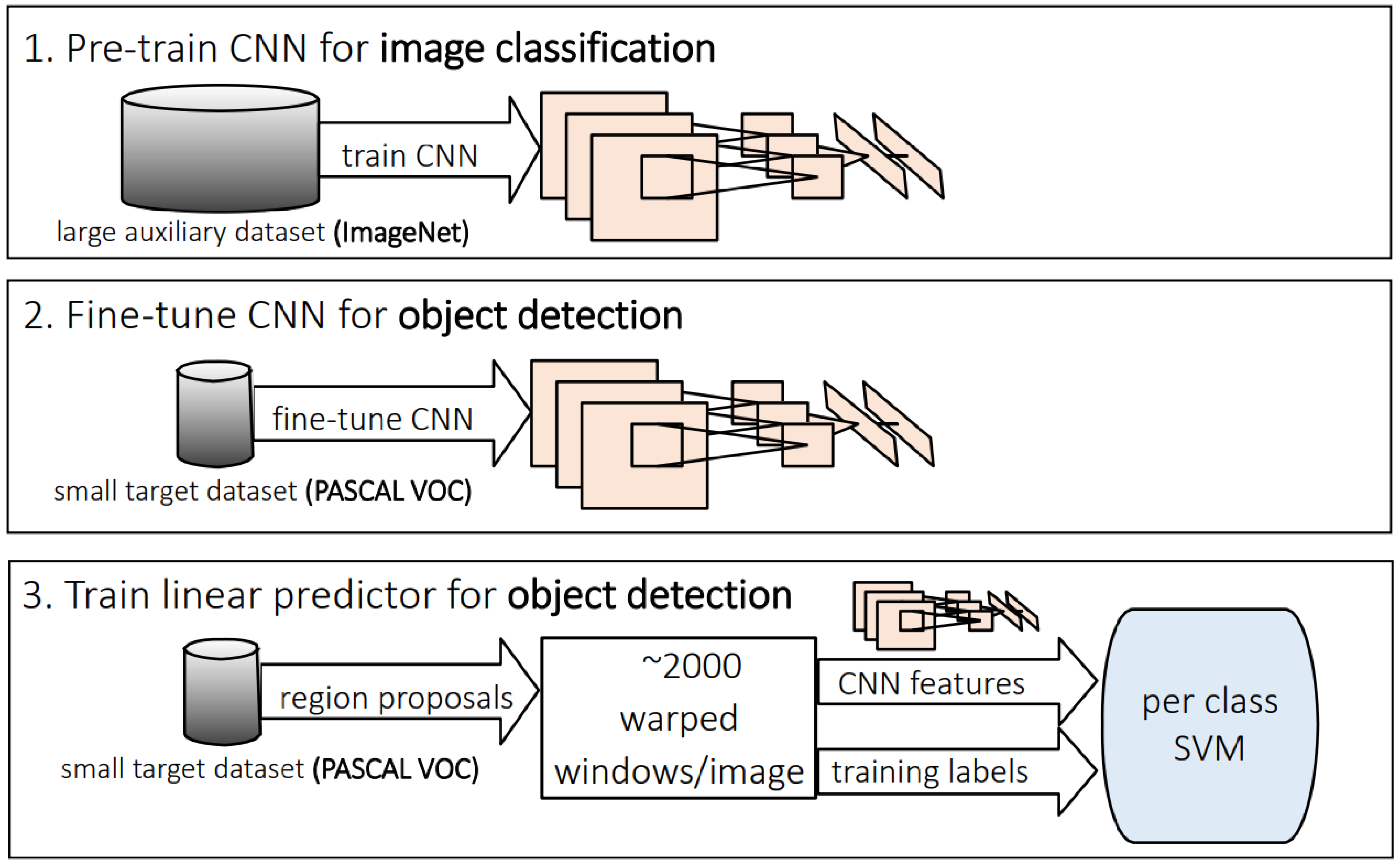
Girshick, Donahue, Darrell and Malik (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR.

R-CNN : Regions with CNN features

- Each region proposal is processed by the CNN, followed by a SVM and a bounding box regressor.



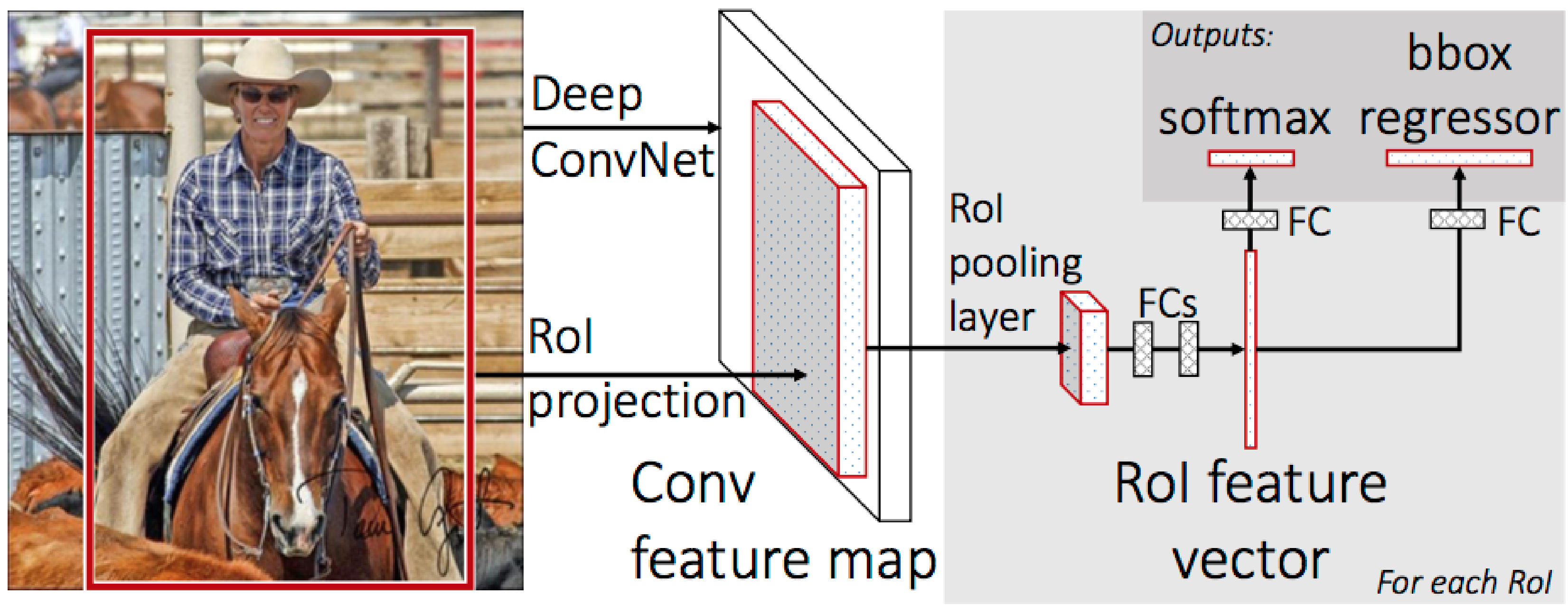
- The CNN is pre-trained on ImageNet and fine-tuned on Pascal VOC.



Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Source:
https://courses.cs.washington.edu/courses/cse590v/14au/cse590v_wk1_rcnn.pdf

Fast R-CNN

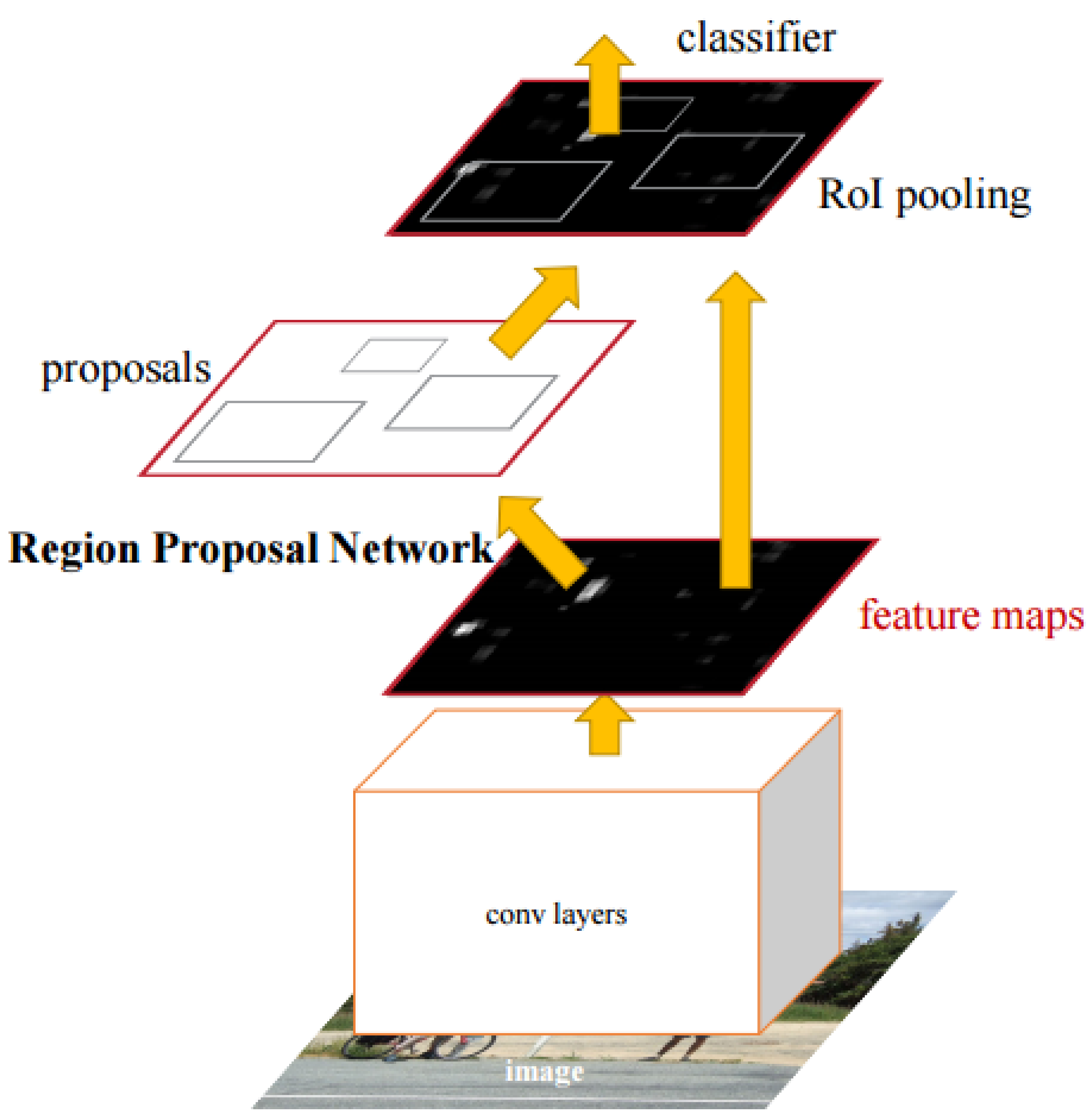


- The main drawback of R-CNN is that each of the 2000 region proposals have to go through the CNN: extremely slow.
- The idea behind **Fast R-CNN** is to extract region proposals in higher feature maps and to use transfer learning.

- The network first processes the whole image with several convolutional and max pooling layers to produce a feature map.
- Each object proposal is projected to the feature map, where a region of interest (RoI) pooling layer extracts a fixed-length feature vector.
- Each feature vector is fed into a sequence of FC layers that finally branch into two sibling output layers:
  - a softmax probability estimate over the K classes plus a catch-all “background” class.
  - a regression layer that outputs four real-valued numbers for each class.
- The loss function to minimize is a composition of different losses and penalty terms:

$$\mathcal{L}(\theta) = \lambda_1 \, \mathcal{L}_\text{classification}(\theta) + \lambda_2 \, \mathcal{L}_\text{regression}(\theta) + \lambda_3 \, \mathcal{L}_\text{regularization}(\theta)$$

Faster R-CNN

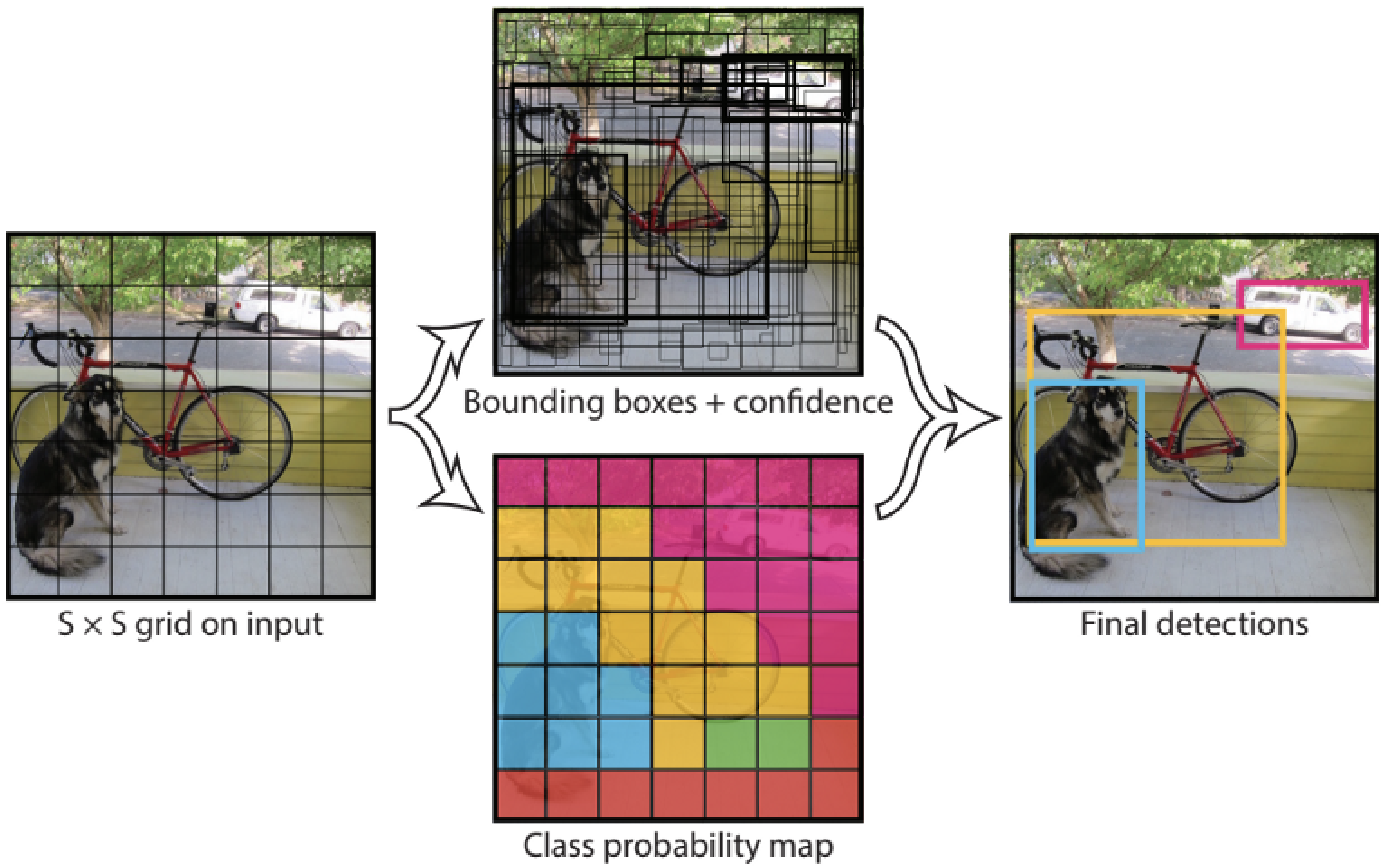


- Both R-CNN and Fast R-CNN use selective search to find out the region proposals: slow and time-consuming.
- Faster R-CNN introduces an object detection algorithm that lets the network learn the region proposals.
- The image is passed through a pretrained CNN to obtain a convolutional feature map.
- A separate network is used to predict the region proposals.
- The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the object and predict the bounding box.

Ren et al. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497

2 - YOLO

YOLO (You Only Look Once)



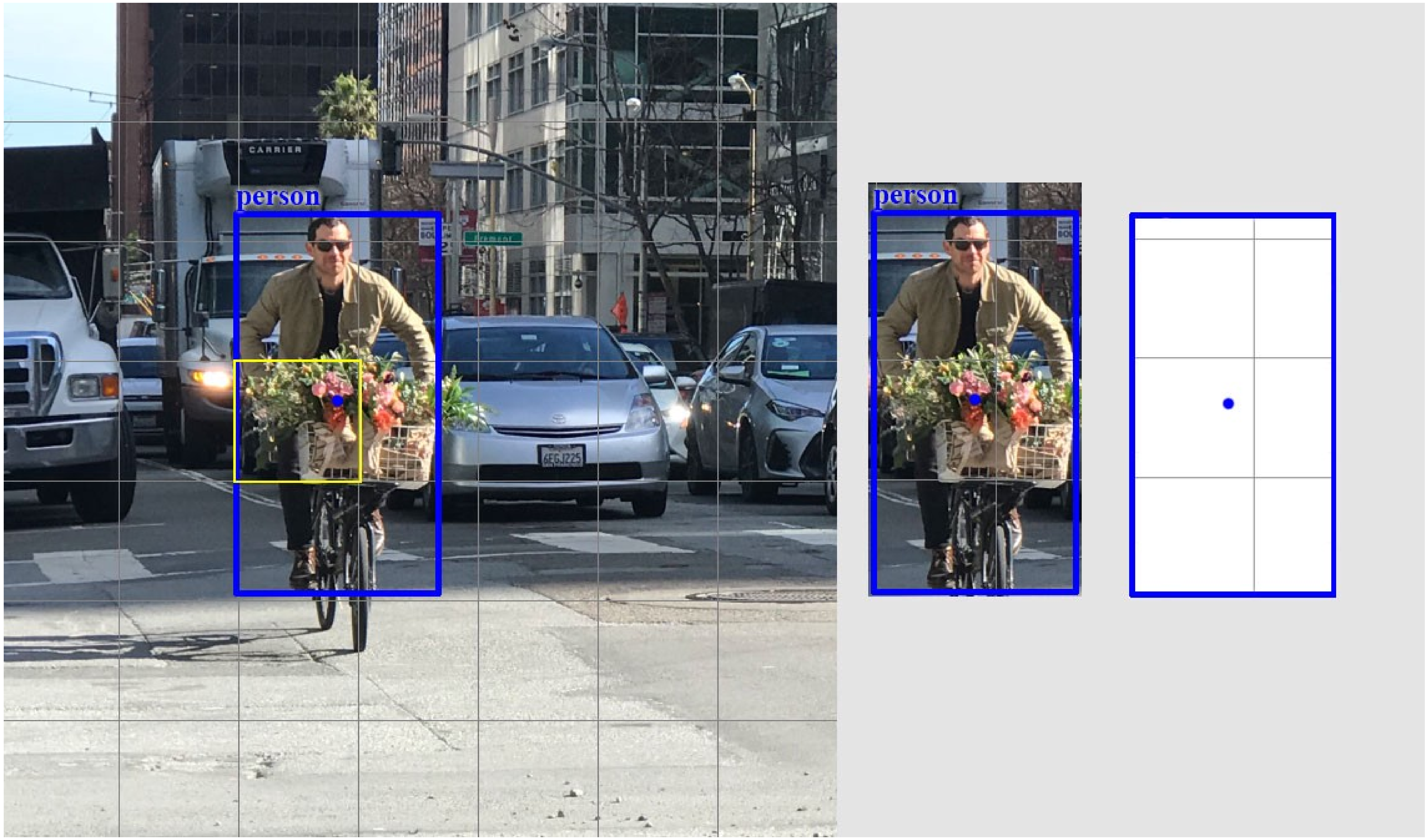
- (Fast(er)) R-CNN perform classification for each region proposal sequentially: slow.
- YOLO applies a single neural network to the full image to predict all possible boxes and the corresponding classes.
- YOLO divides the image into a SxS grid of cells.

- Each grid cell predicts a single object, with the corresponding C **class probabilities** (softmax).
- It also predicts the coordinates of B possible **bounding boxes** (x, y, w, h) as well as a box **confidence score**.
- The SxSxB predicted boxes are then pooled together to form the final prediction.

Redmon et al. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv:1506.02640

YOLO (You Only Look Once)

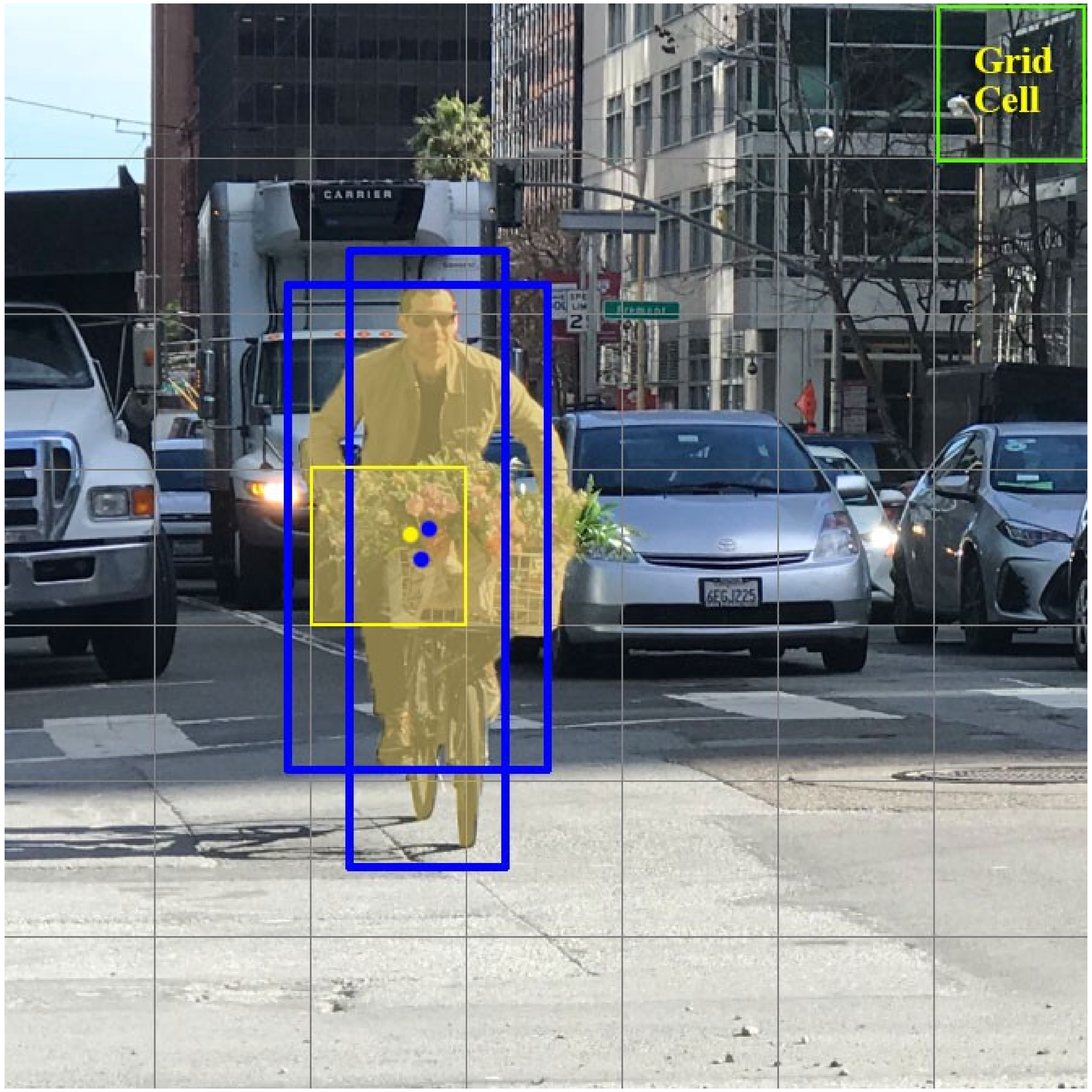
- The yellow box predicts the presence of a **person** (the class) as well as a candidate **bounding box** (it may be bigger than the grid cell itself).



Source: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

YOLO (You Only Look Once)

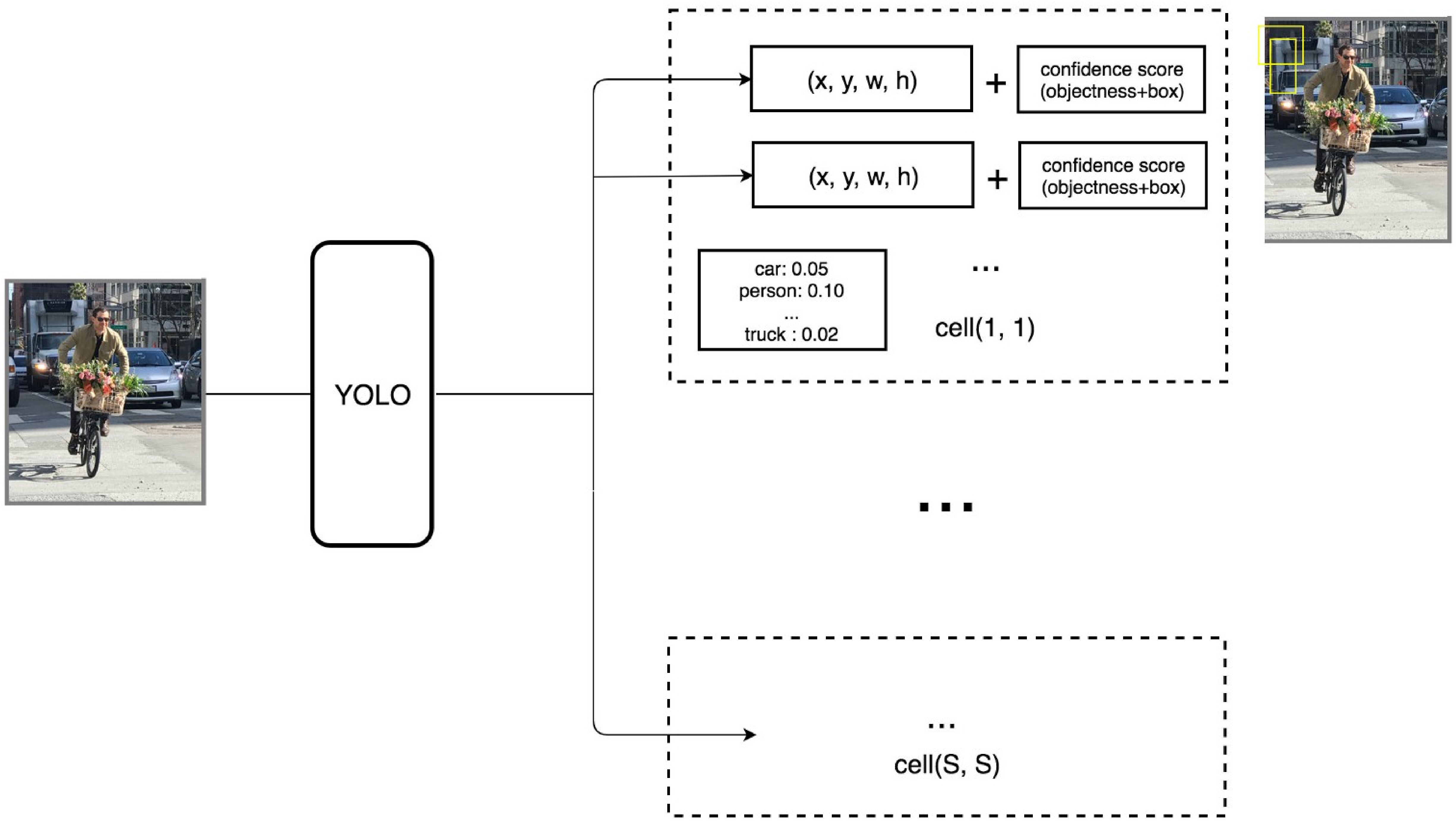
- We will suppose here that each grid cell proposes 2 bounding boxes.



Source: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

YOLO (You Only Look Once)

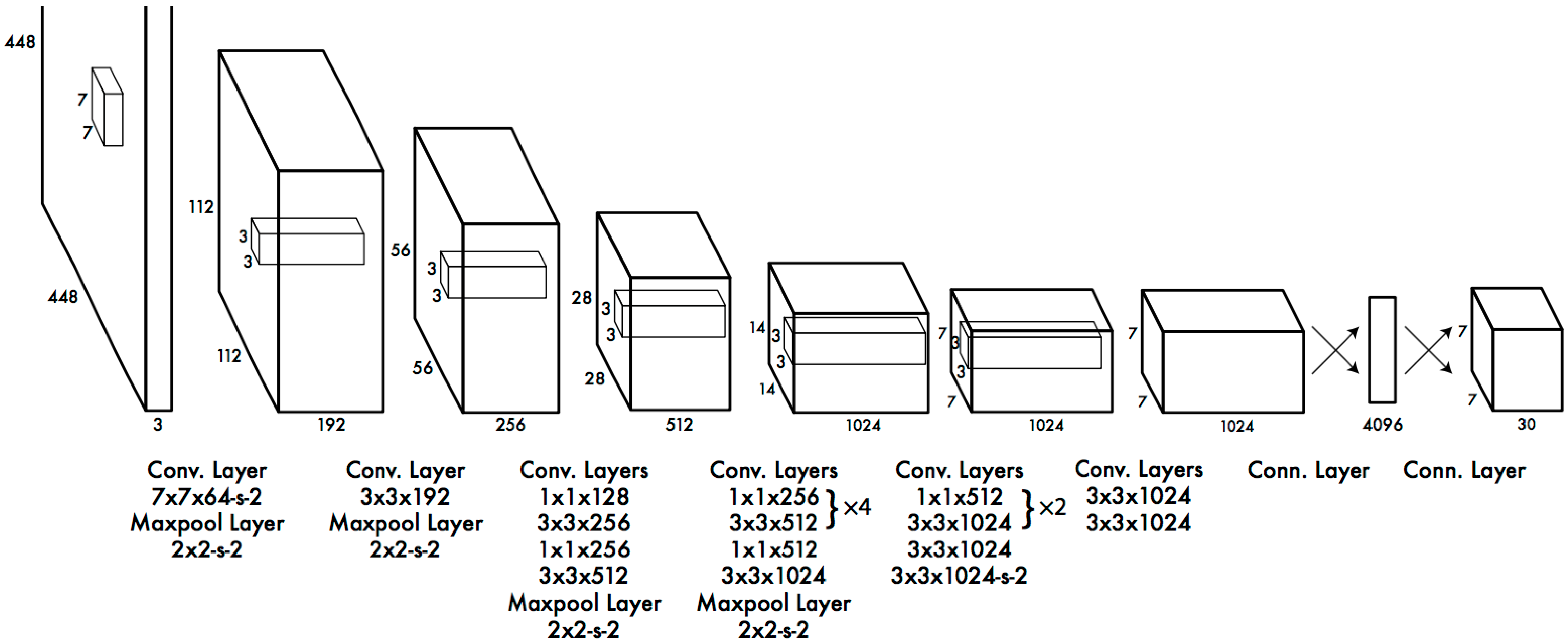
- Each grid cell predicts a probability for each of the 20 classes, and two bounding boxes (4 coordinates and a confidence score per bounding box).
- This makes C + B * 5 = 30 values to predict for each cell.



Source: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

YOLO : CNN architecture

- YOLO uses a CNN with 24 convolutional layers and 4 max-pooling layers to obtain a 7x7 grid.
- The last convolution layer outputs a tensor with shape (7, 7, 1024). The tensor is then flattened and passed through 2 fully connected layers.
- The output is a tensor of shape (7, 7, 30), i.e. 7x7 grid cells, 20 classes and 2 boundary box predictions per cell.



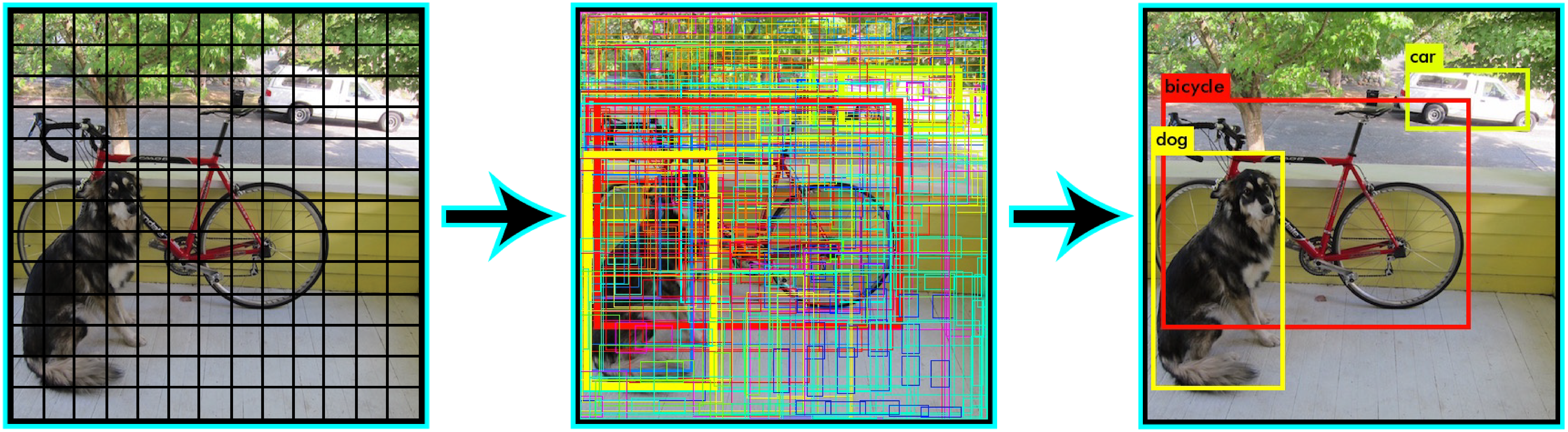
Redmon et al. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv:1506.02640

YOLO : confidence score

- The 7x7 grid cells predict 2 bounding boxes each: maximum of 98 bounding boxes on the whole image.
- Only the bounding boxes with the **highest class confidence score** are kept.

$$\text{class confidence score} = \text{box confidence score} * \text{class probability}$$

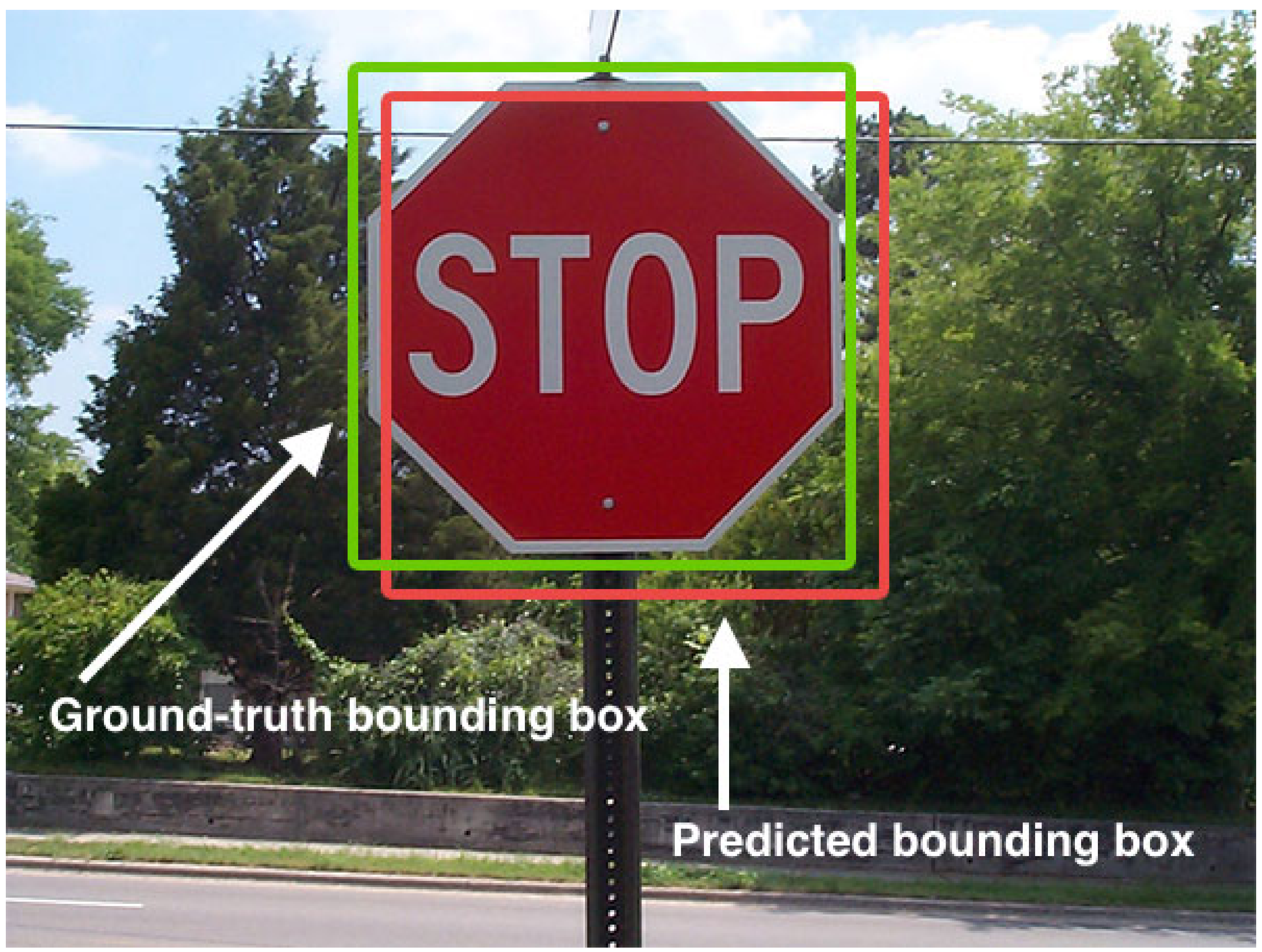
- In practice, the class confidence score should be above 0.25 to be retained.

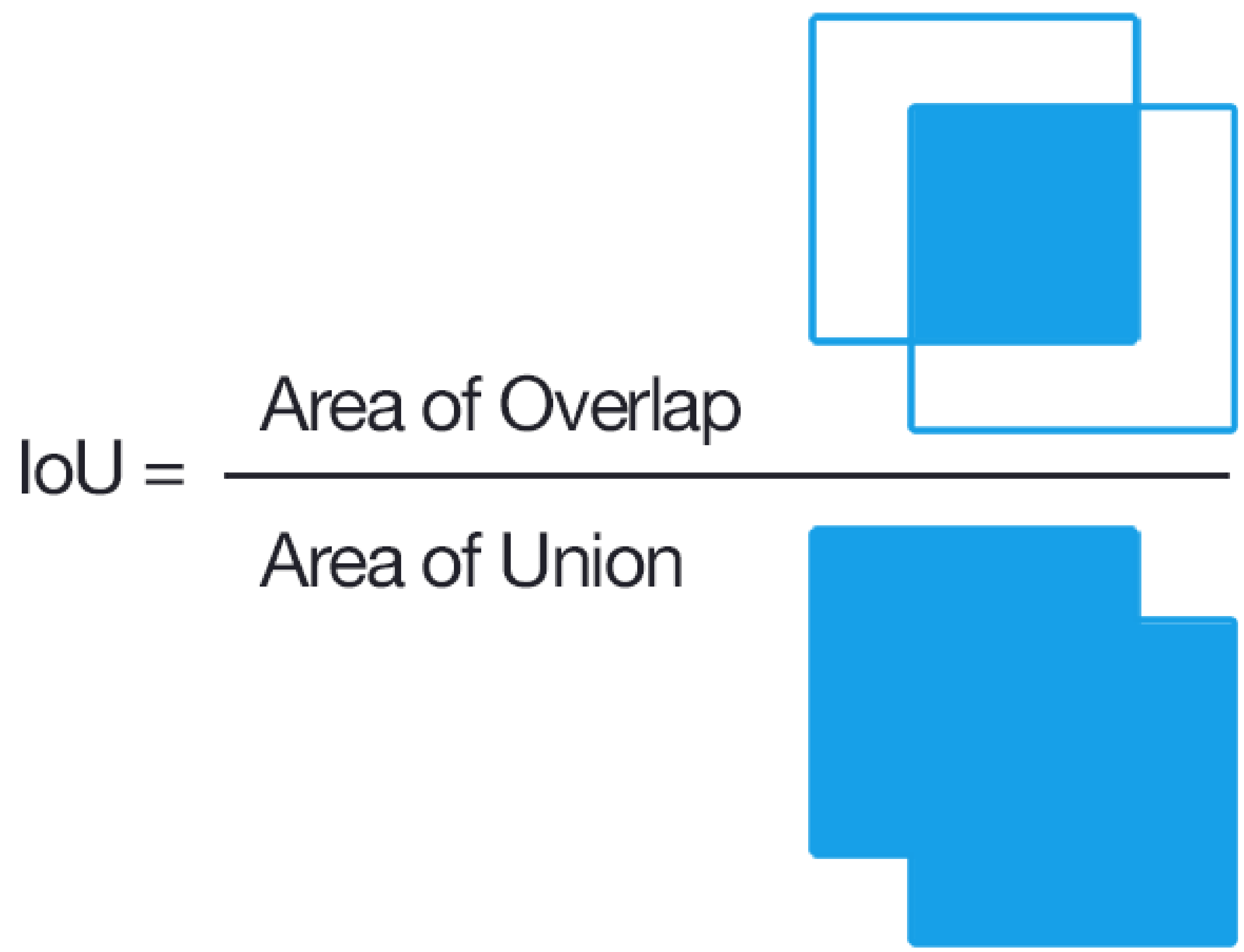


Redmon et al. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv:1506.02640

YOLO : Intersection over Union (IoU)

- To ensure specialization, only one bounding box per grid cell should be responsible for detecting an object.
- During learning, we select the bounding box with the biggest overlap with the object.
- This can be measured by the **Intersection over the Union** (IoU).





Source: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

YOLO : loss functions

- The output of the network is a 7x7x30 tensor, representing for each cell:
    - the probability that an object of a given class is present.
    - the position of two bounding boxes.
    - the confidence that the proposed bounding boxes correspond to a real object (the IoU).
- We are going to combine three different loss functions:

1. The **categorization loss**: each cell should predict the correct class.
2. The **localization loss**: error between the predicted boundary box and the ground truth for each object.
3. The **confidence loss**: do the predicted bounding boxes correspond to real objects?

Redmon et al. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv:1506.02640

YOLO : classification loss

- The classification loss is the **mse** between:
  - $\hat{p}_i(c)$: the one-hot encoded class c of the object present under each cell i, and
  - $p_i(c)$: the predicted class probabilities of cell i.

$$\mathcal{L}_{\text{classification}}(\theta) = \sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

where 1_i^{obj} is 1 when there actually is an object behind the cell i, 0 otherwise (background).

- They could also have used the cross-entropy loss, but the output layer is not a softmax layer.
- Using mse is also more compatible with the other losses.

YOLO : localization loss

- For all bounding boxes matching a real object, we want to minimize the **mse** between:
  - $(\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)$: the coordinates of the ground truth bounding box, and
  - (x_i, y_i, w_i, h_i): the coordinates of the predicted bounding box.

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

where 1_{ij}^obj is 1 when the bounding box j of cell i “matches” with an object (IoU).

- The root square of the width and height of the bounding boxes is used.
- This allows to penalize more the errors on small boxes than on big boxes.

YOLO : confidence loss

- Finally, we need to learn the confidence score of each bounding box, by minimizing the **mse** between:
  - C_i: the predicted confidence score of cell i, and
  - $\hat{C}_i$: the IoU between the ground truth bounding box and the predicted one.

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

- Two cases are considered:
  1. There was a real object at that location ($1_{ij}^\text{obj} = 1$): the confidences should be updated fully.
  2. There was no real object ($1_{ij}^\text{noobj} = 1$): the confidences should only be moderately updated ($\lambda^\text{noobj} = 0.5$)
- This is to deal with **class imbalance**: there are much more cells on the background than on real objects.

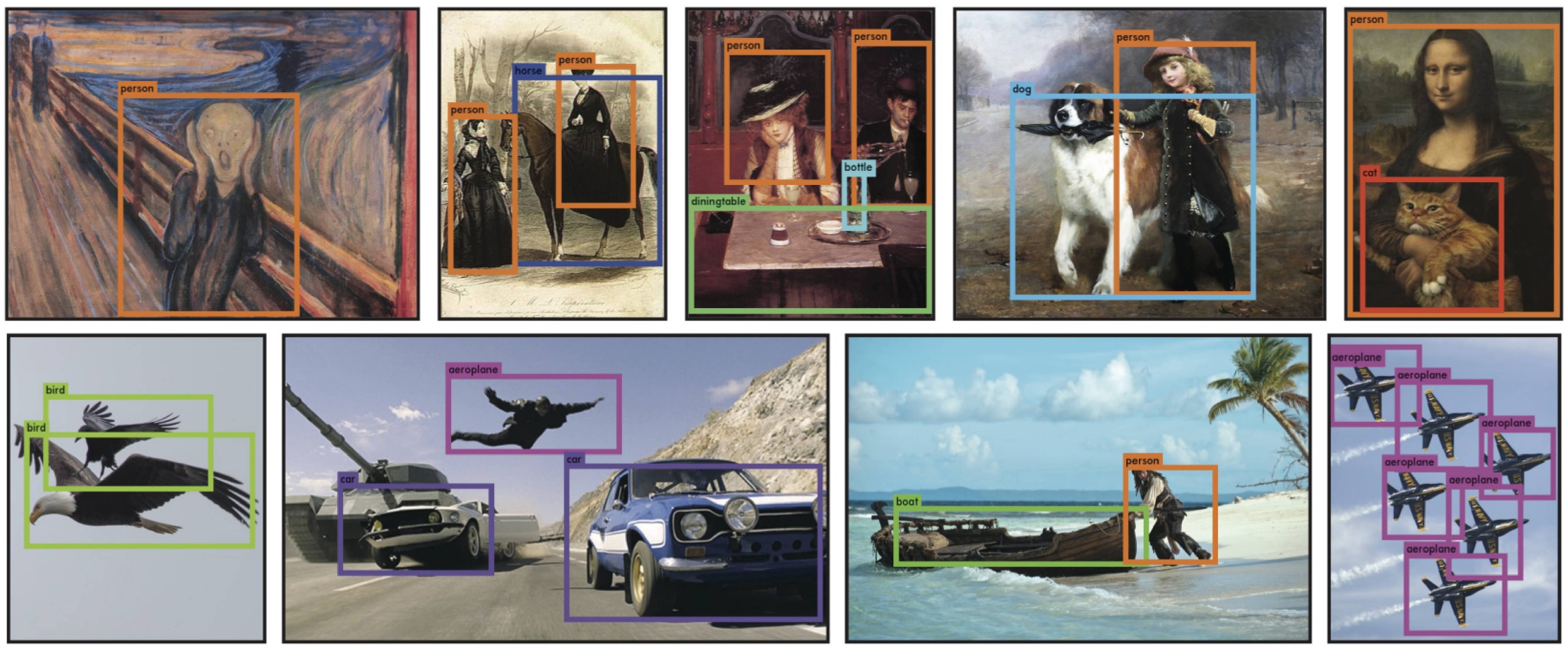
YOLO : loss function

- Put together, the loss function to minimize is:

$$
\begin{aligned}
\mathcal{L}(\theta) &= \mathcal{L}_{\text{classification}}(\theta) + \lambda_{\text{coord}} \mathcal{L}_{\text{localization}}(\theta) + \mathcal{L}_{\text{confidence}}(\theta) && (1) \\
&= \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 && (2) \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] && (3) \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] && (4) \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_{ij} - \hat{C}_{ij})^2 && (5) \\
&+ \lambda^{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2 && (6)
\end{aligned}
$$

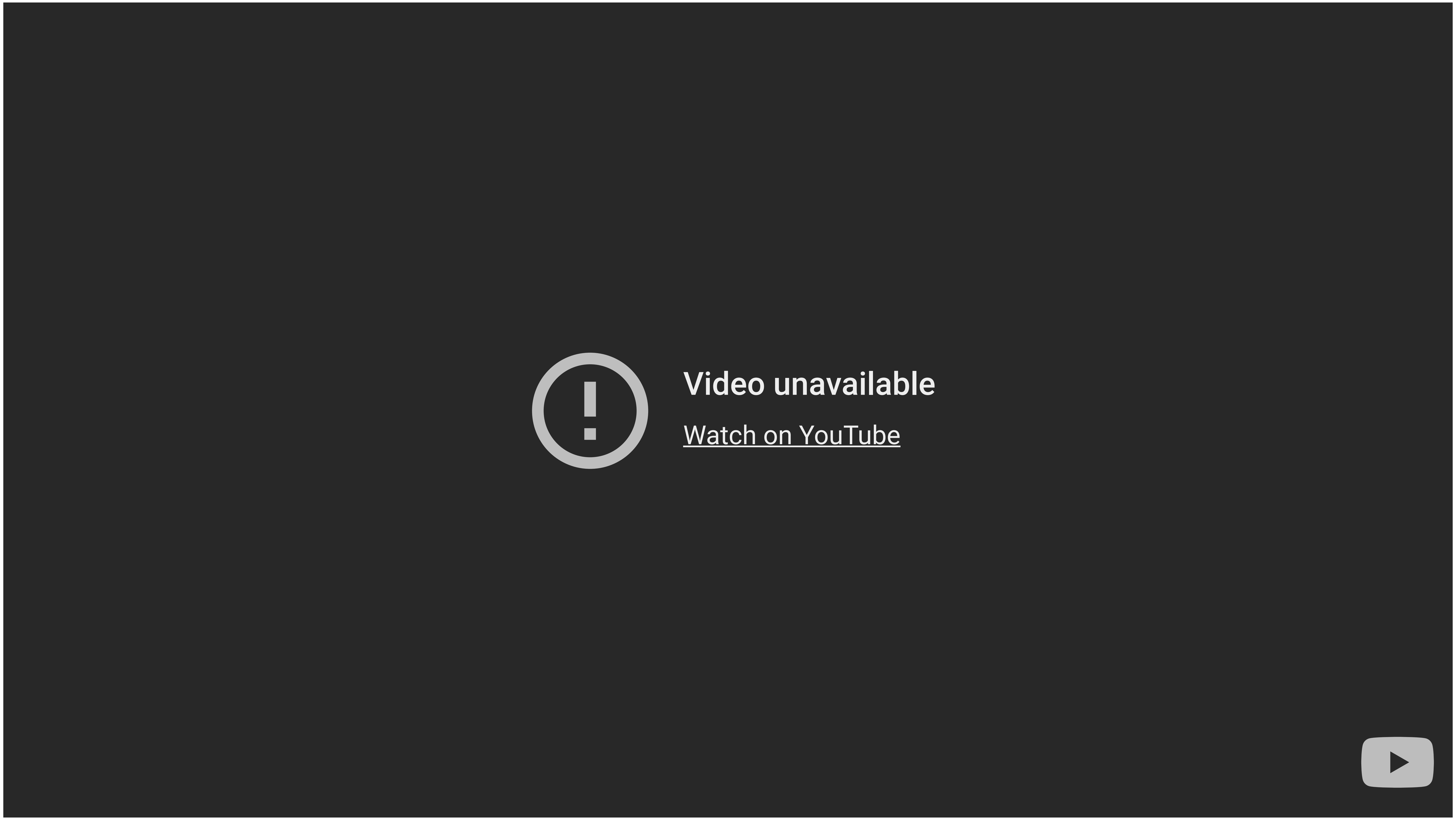
Redmon et al. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv:1506.02640

YOLO : Training on PASCAL VOC



	VOC 2007 AP	Picasso AP	Picasso Best F_1	People-Art AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

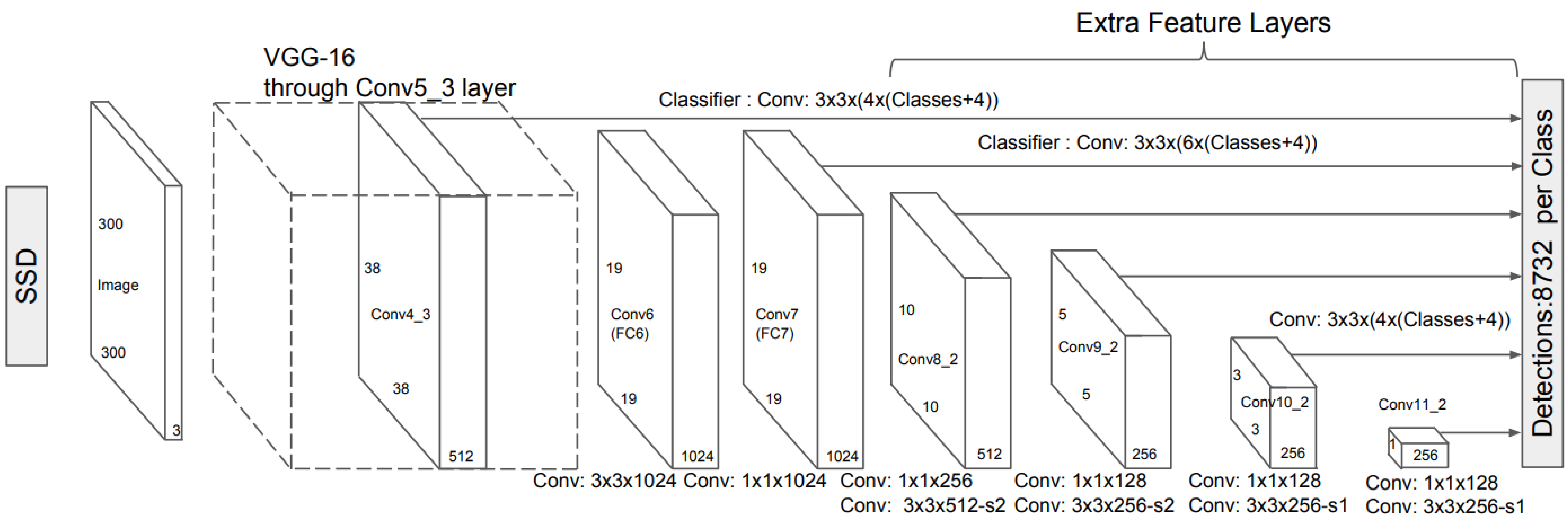
- YOLO was trained on PASCAL VOC (natural images) but generalizes well to other datasets (paintings...).
- Runs real-time (60 fps) on a NVIDIA Titan X.
- Faster and more accurate versions of YOLO have been developed: YOLO9000, YOLOv3, YOLOv4, YOLOv5...



https://pjreddie.com/darknet/yolo/

3 - Other object detectors

SSD: Single-Shot Detector

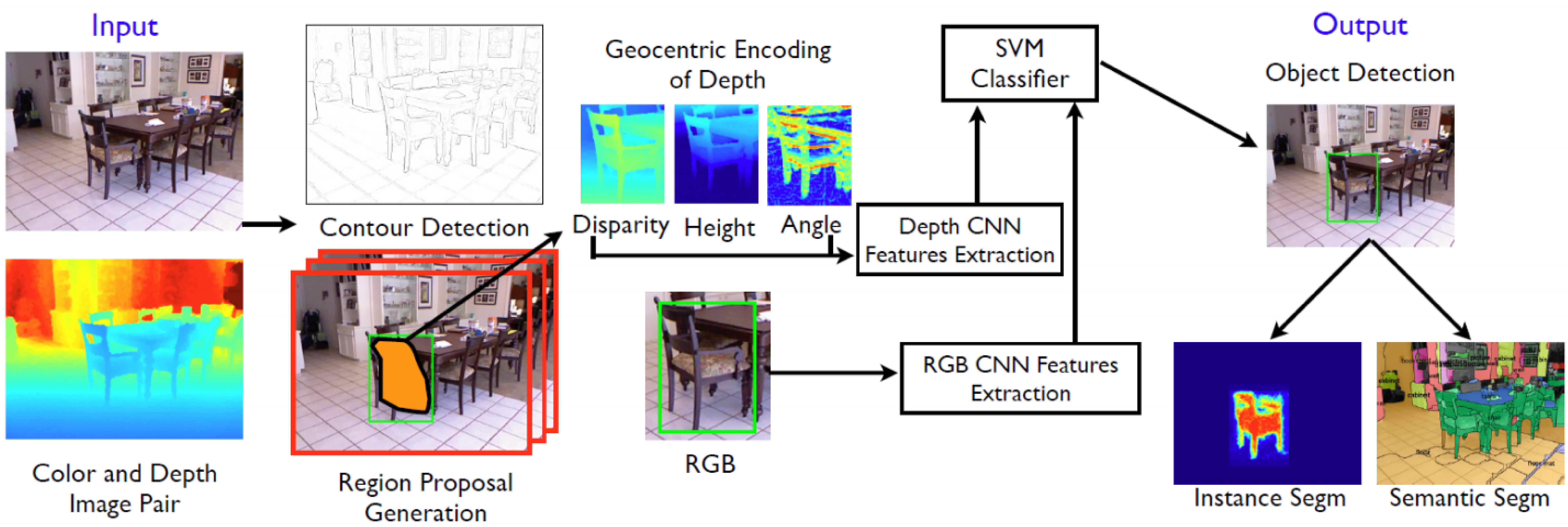


- The idea of SSD is similar to YOLO, but:
  - faster
  - more accurate
  - not limited to 98 objects per scene
  - multi-scale
- Contrary to YOLO, all convolutional layers are used to predict a bounding box, not just the final tensor.
  - Skip connections.
- This allows to detect boxes at multiple scales (pyramid).

Liu et al. (2016) SSD: Single Shot MultiBox Detector. arXiv:1512.02325

R-CNNs on RGB-D images

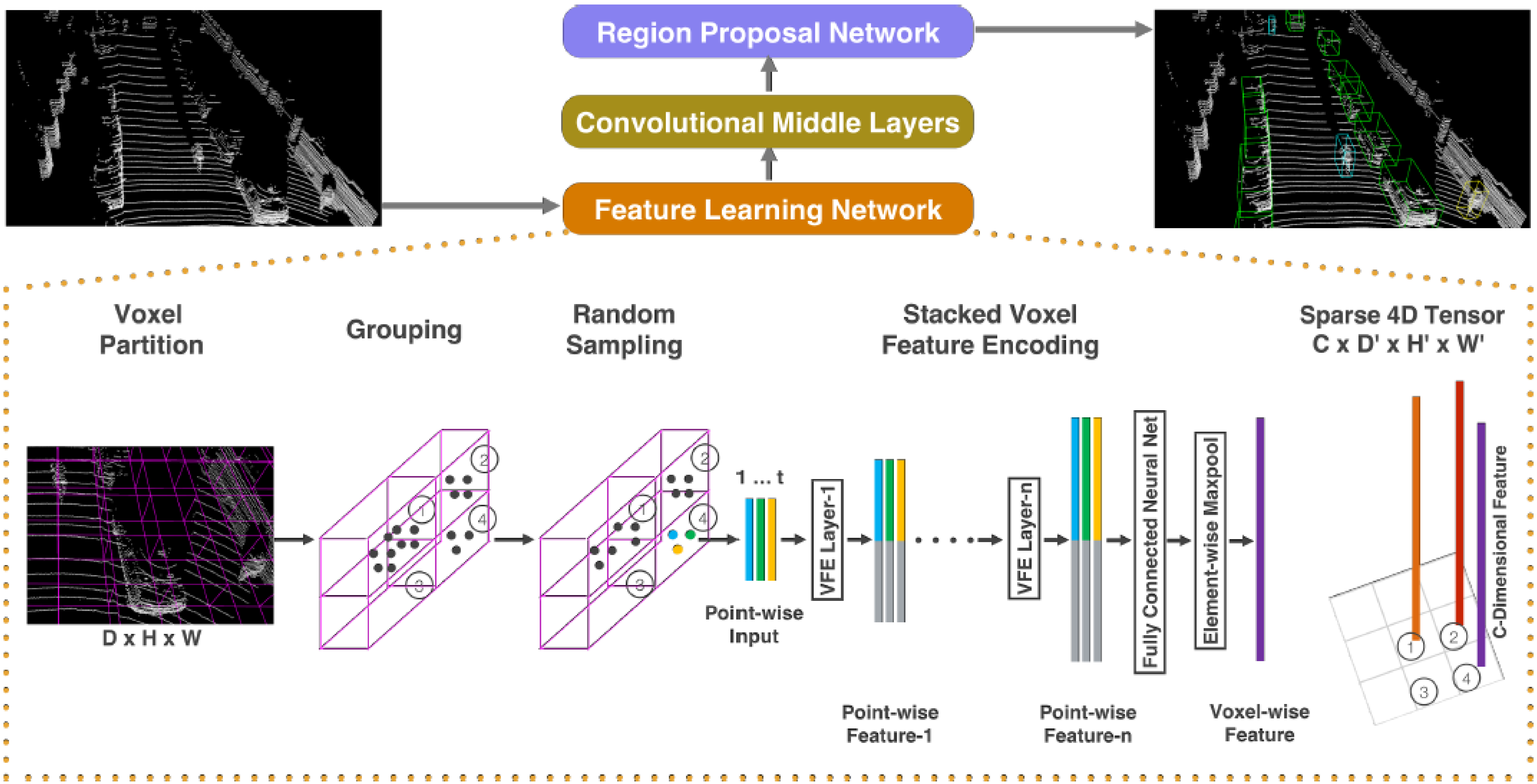
- It is also possible to use **depth** information (e.g. from a Kinect) as an additional channel of the R-CNN.
- The depth information provides more information on the structure of the object, allowing to disambiguate certain situations (segmentation).



Gupta et al. (2014). Learning Rich Features from RGB-D Images for Object Detection and Segmentation, ECCV 2014.

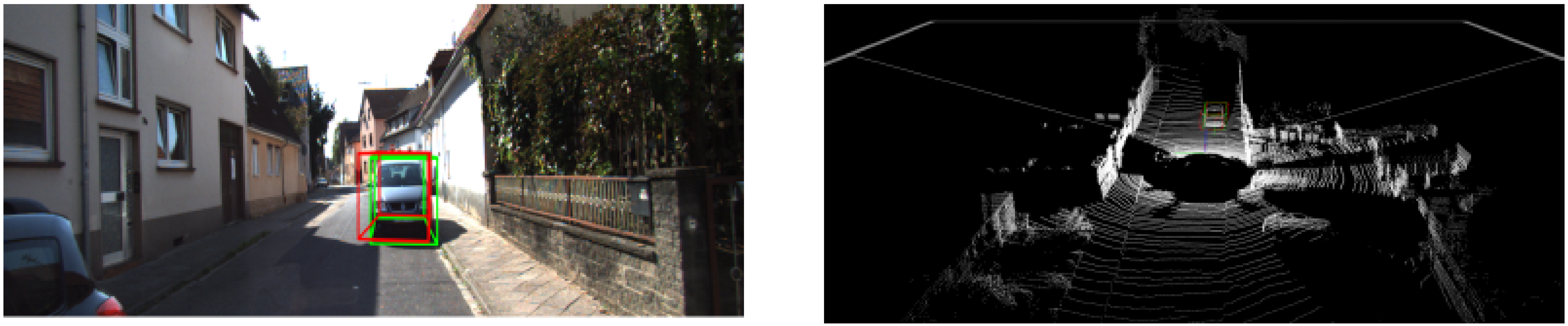
VoxelNet

- Lidar point clouds can also be used for detecting objects, for example **VoxelNet** trained on the KITTI dataset.



Zhou Y, Tuzel O. (2017). VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. arXiv:171106396

VoxelNet



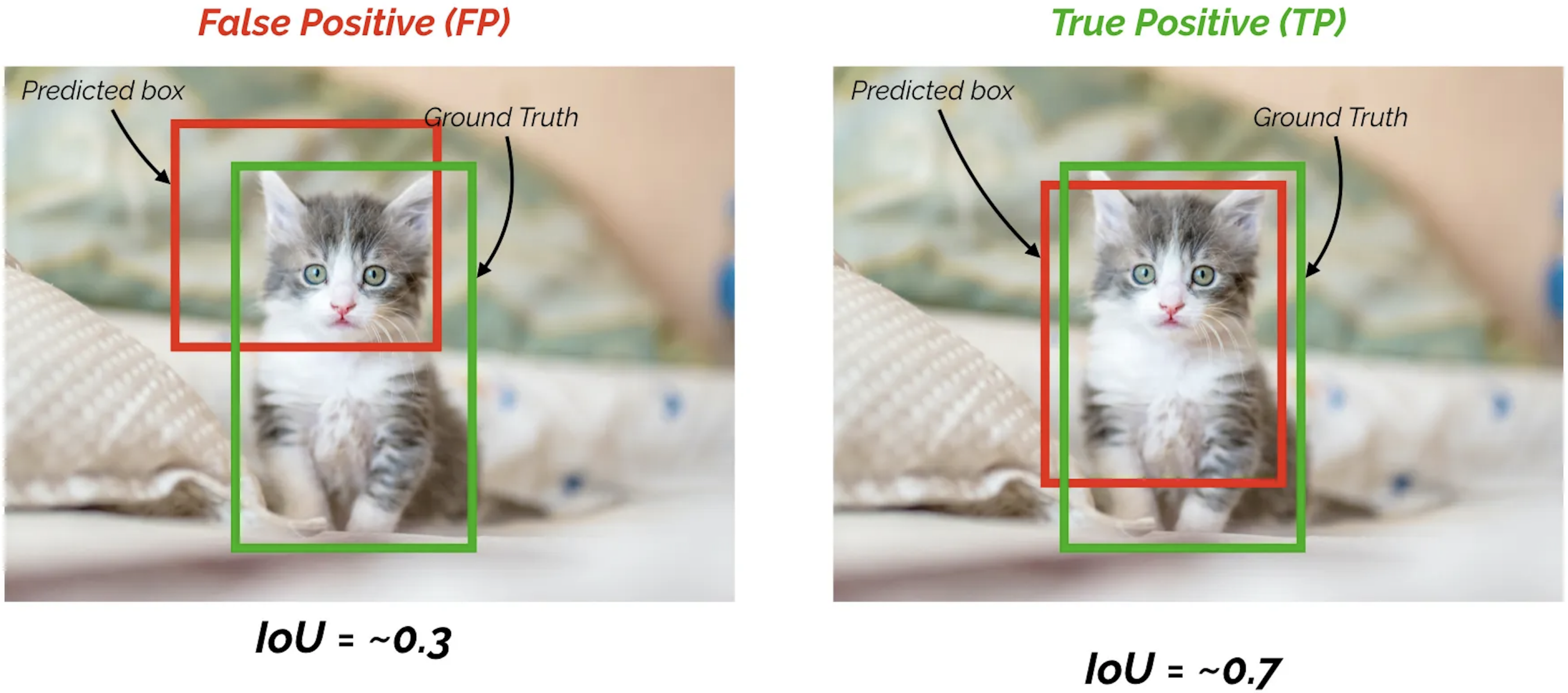
(a) (b)

Source: https://medium.com/@SmartLabAI/3d-object-detection-from-lidar-data-with-deep-learning-95f6d400399a

4 - Metrics

Metrics for object detection

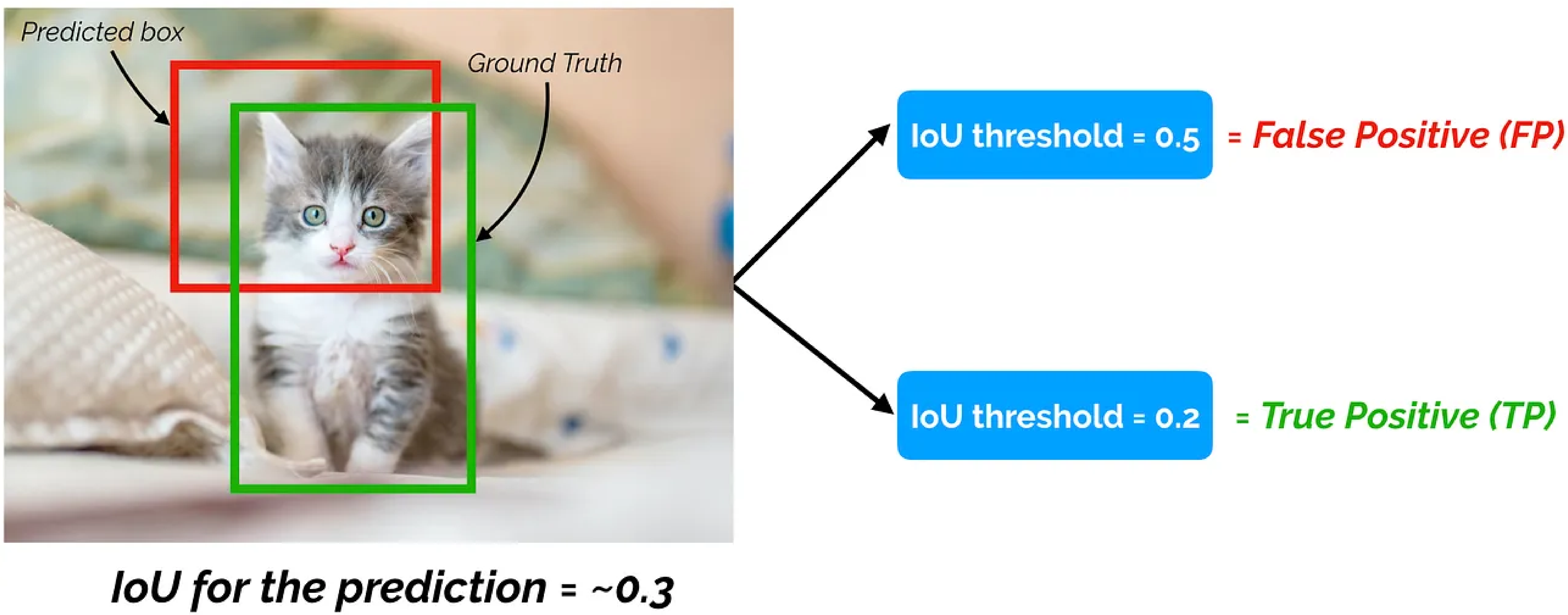
- How do we measure the “accuracy” of an object detector? The output is both a classification and a regression.
- Not only must the predicted class be correct, but the predicted bounding box must overlap with the ground truth, i.e. have an high IoU.



Source: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

Metrics for object detection

- The accuracy of an object detector depends on a threshold for the IoU, for example 0.5.
- A prediction is correct if the predicted class is correct **and** the IoU is above the threshold.



Source: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

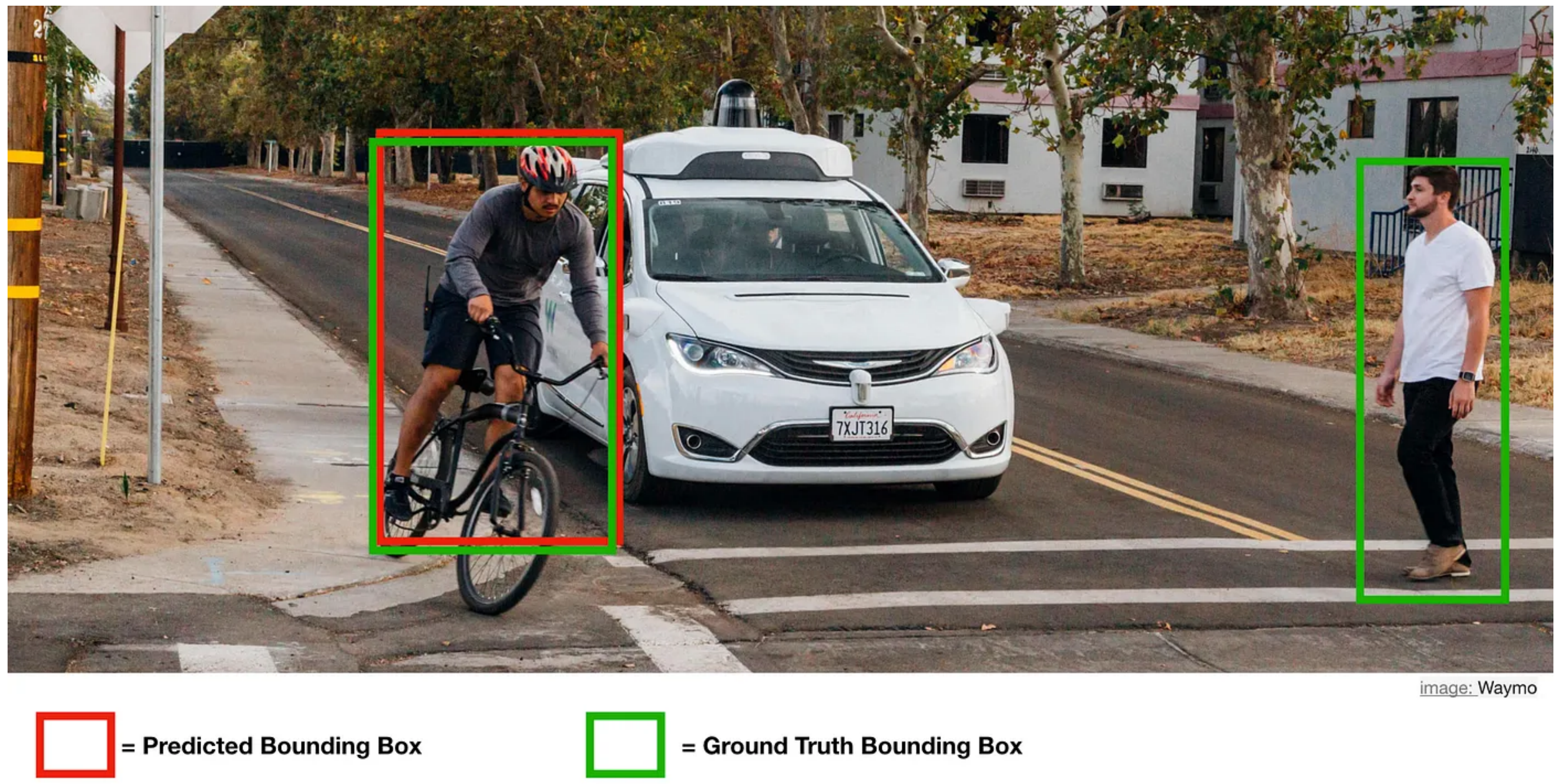
Precision and recall

- For a given class (e.g. “human”), we can compute the binary **precision** and **recall** values:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- P = when something is detected, is it correct? R = if something exists, is it detected?
- In the image below, we have one TP, one FN, zero FP and an undefined number of TN:

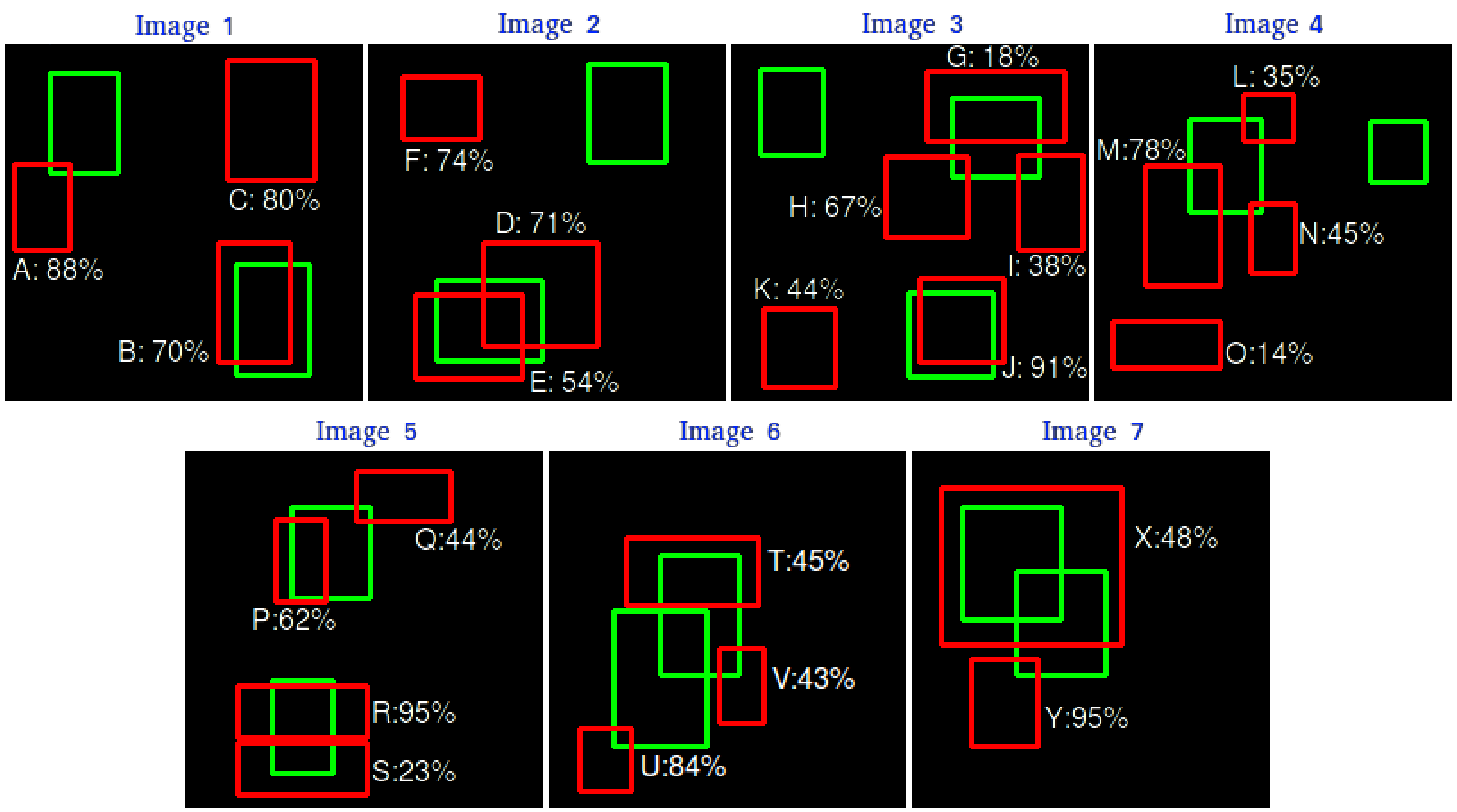
$$P = \frac{1}{1+0} = 1 \quad R = \frac{1}{1+1} = 0.5$$



Source: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

mAP: mean average precision

- Let's now compute the **precision-recall curve** over 7 images, with 15 ground truth boxes and 24 predictions.



Source: https://github.com/rafaelpadilla/Object-Detection-Metrics

- Each prediction has a confidence score for the classification, and is either a TP or FP (depending on the IoU threshold).

Images	Detections	Confidences	TP or FP
Image 1	A	88%	FP
Image 1	B	70%	TP
Image 1	C	80%	FP
Image 2	D	71%	FP
Image 2	E	54%	TP
Image 2	F	74%	FP
Image 3	G	18%	TP
Image 3	H	67%	FP
Image 3	I	38%	FP
Image 3	J	91%	TP
Image 3	K	44%	FP
Image 4	L	35%	FP
Image 4	M	78%	FP
Image 4	N	45%	FP
Image 4	O	14%	FP
Image 5	P	62%	TP
Image 5	Q	44%	FP
Image 5	R	95%	TP
Image 5	S	23%	FP
Image 6	T	45%	FP
Image 6	U	84%	FP
Image 6	V	43%	FP
Image 7	X	48%	TP
Image 7	Y	95%	FP

mAP: mean average precision

- Let's now **sort** the predictions with a decreasing confidence score and **incrementally** compute the prediction and recall:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

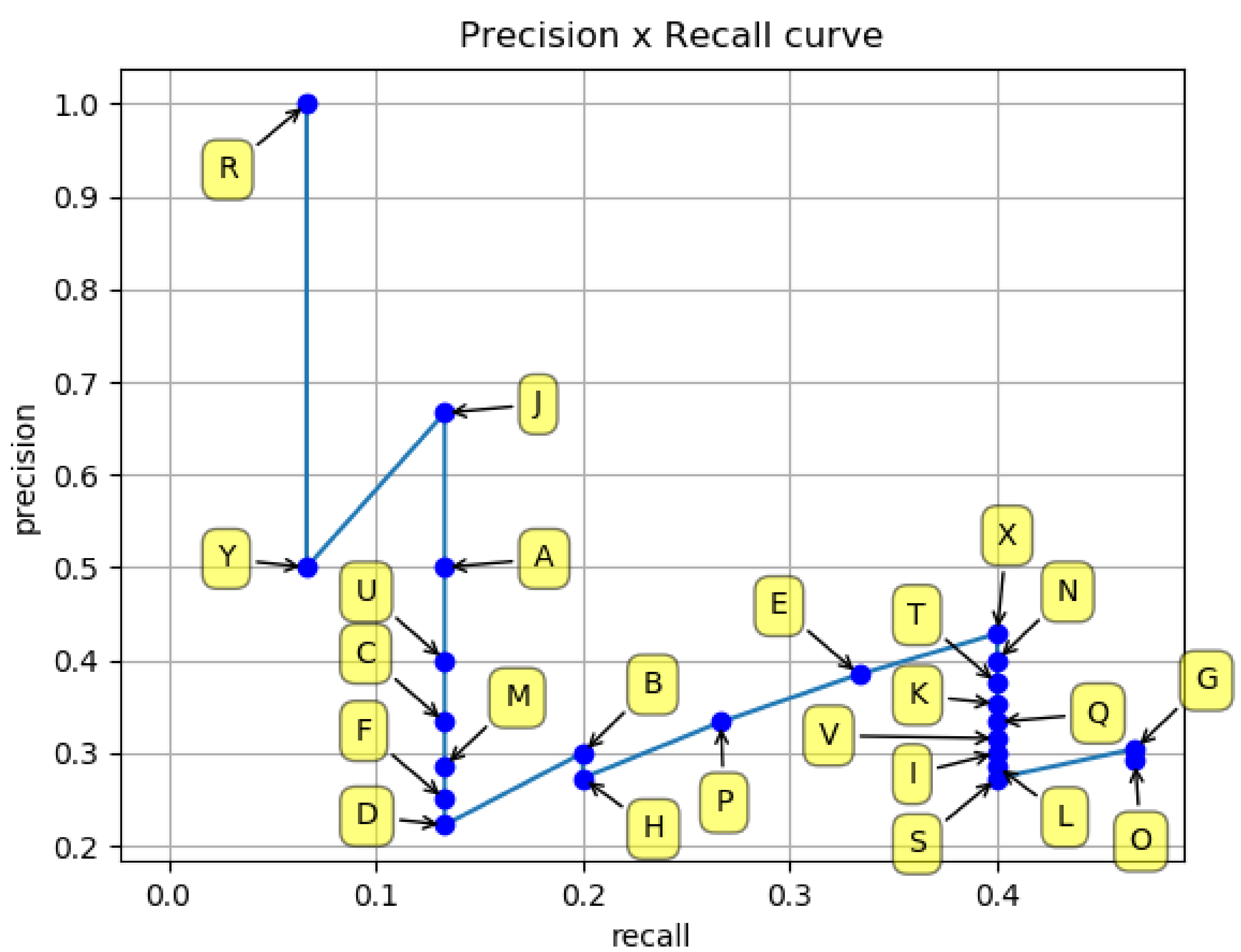
- We just accumulate the number of TP and FP over the 24 predictions.
- Note that TP + FN is the number of ground truths and is constant (15), so the recall will increase.
- This equivalent to setting a high threshold for the confidence score and progressively decreasing it.

Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
Image 4	O	14%	0	1	7	17	0.2916	0.4666

Source: https://github.com/rafaelpadilla/Object-Detection-Metrics

mAP: mean average precision

- If we plot the **precision x recall curve** (PR curve) for the 24 predictions, we obtain:



- The precision globally decreases with the recall, as we use predictions with lower confidence scores, but there are some oscillations.

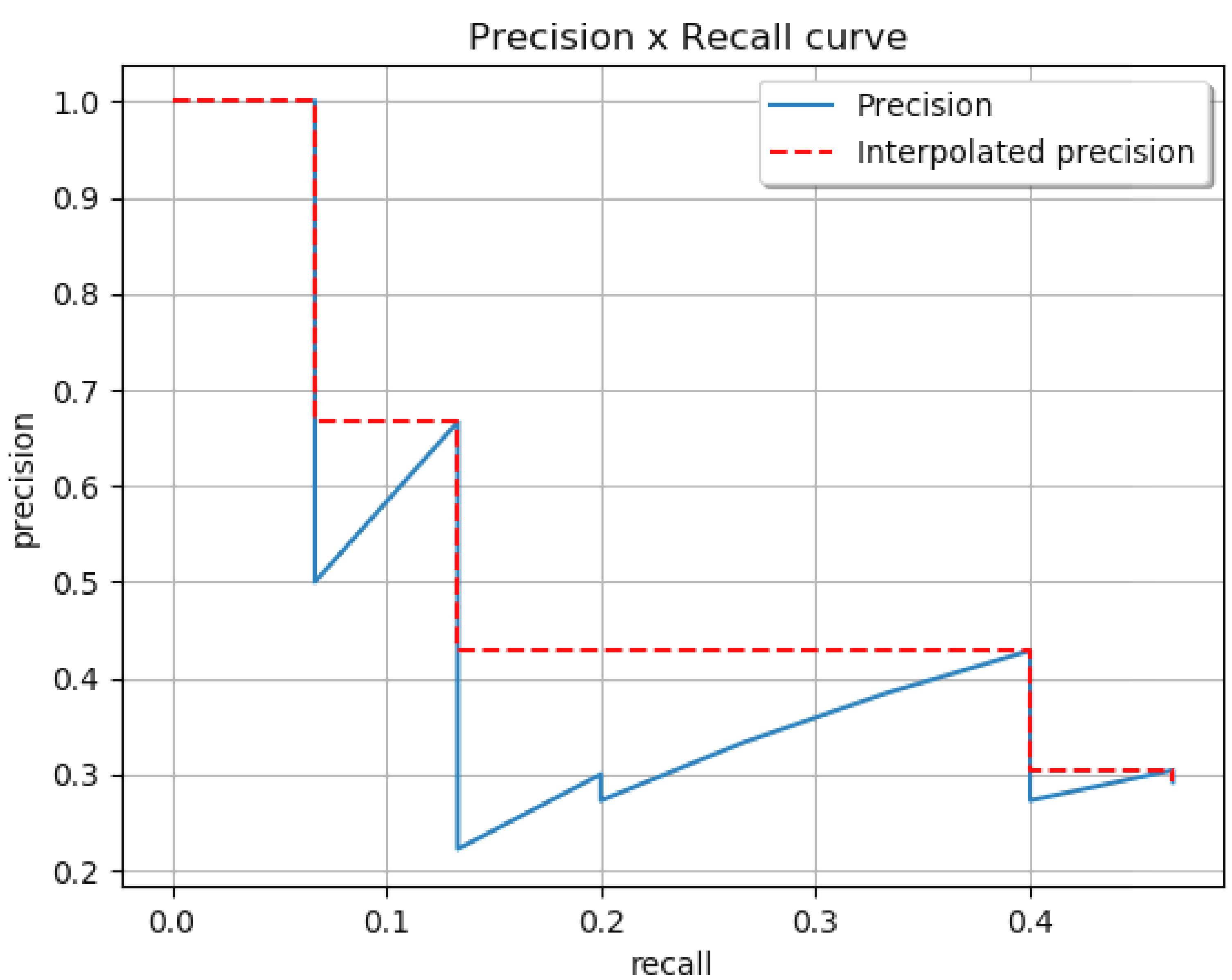
Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
Image 4	O	14%	0	1	7	17	0.2916	0.4666

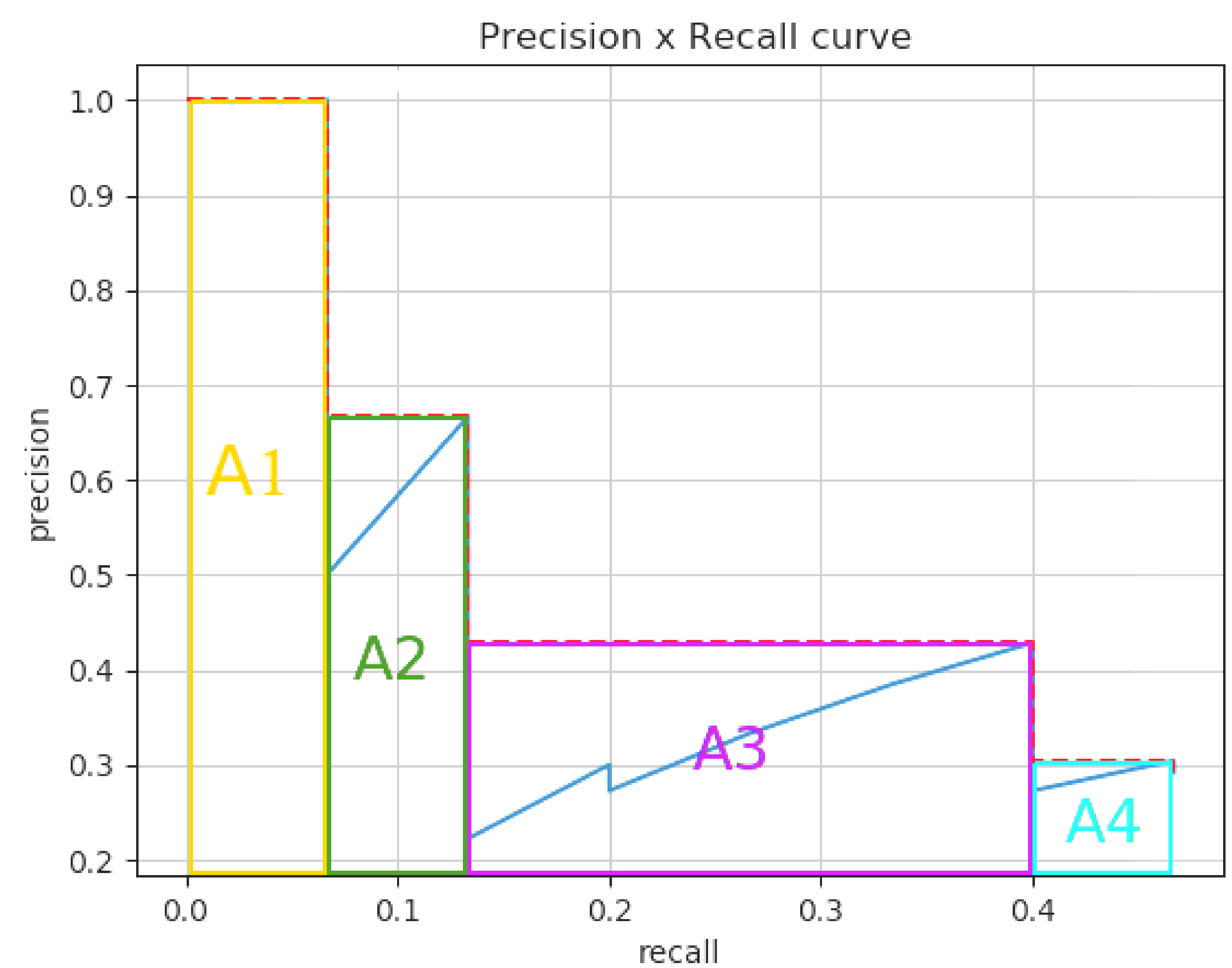
Source: https://github.com/rafaelpadilla/Object-Detection-Metrics

mAP: mean average precision

- To get rid of these oscillations, we **interpolate** the precision by taking maximal precision value for higher recall (left).
- We can then easily integrate this curve by computing the **area under the curve** (AUC, right), what defines the **average precision** (AP).

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





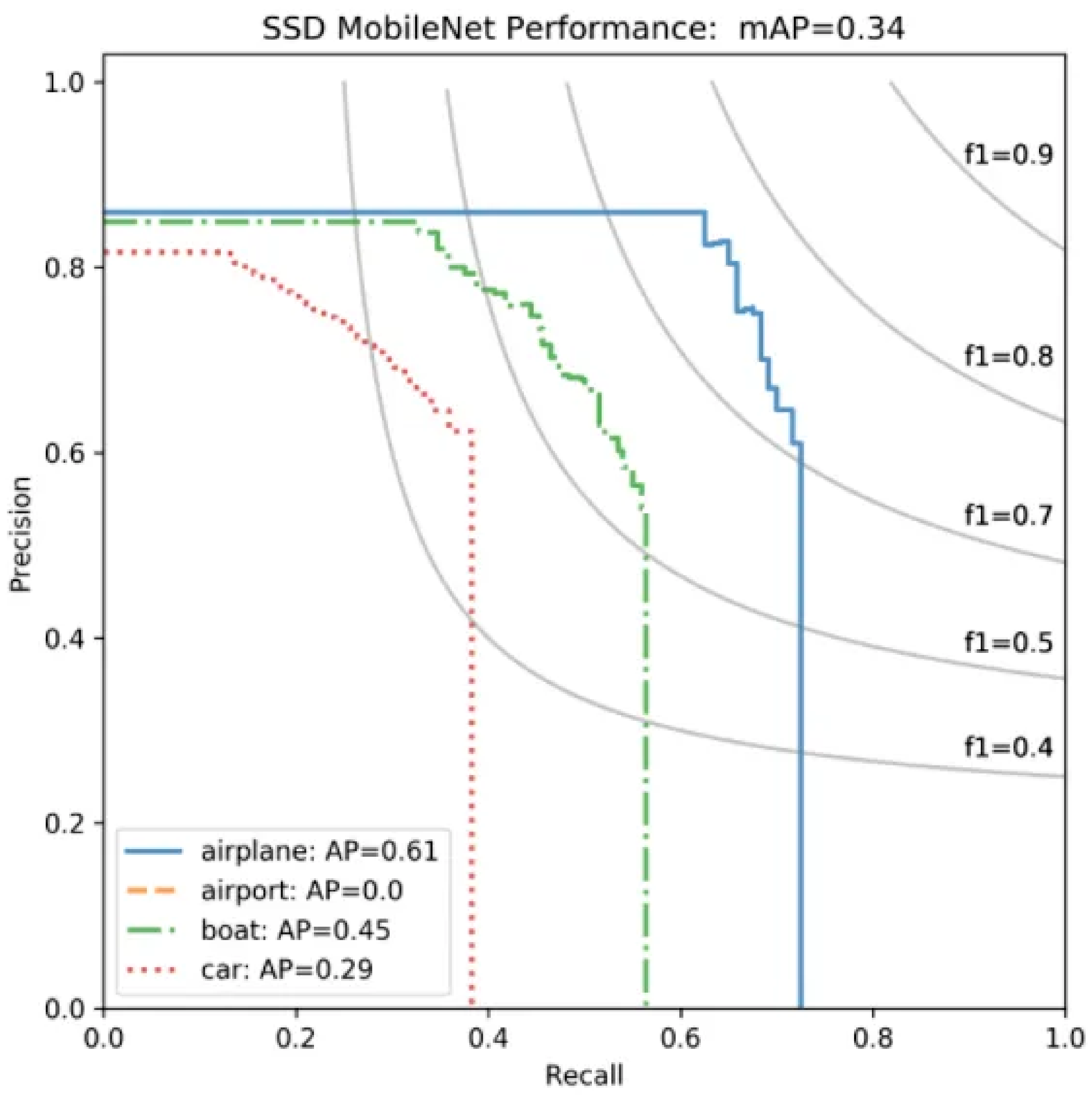
Source: https://github.com/rafaelpadilla/Object-Detection-Metrics

mAP: mean average precision

- A good detector sees its precision decreases not that much when the recall increases, i.e. when it is still correct when it increasingly detects objects.
- The ideal detector has an AP of 1.
- When averaging the AP over the classes, one obtains the **mean average precision (mAP)**:

$$\text{mAP} = \frac{1}{N_\text{classes}} \sum_{i=1}^{N_\text{classes}} AP_i$$

- One usually reports the mAP value with the IoU threshold, e.g. `mAP@0.5`.
- mAP is a better trade-off between precision and recall than the F1 score.
- `scikit-learn` is your friend:



Source: Van Etten, A. (2019). Satellite Imagery Multiscale Rapid Detection with Windowed Networks. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), 735–743. doi:10.1109/WACV.2019.00083

```python
mAP = sklearn.metrics.average_precision_score(t, y, average="micro")
```

Additional resources on object detection

- https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852
- https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8
- https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088
- https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
- https://towardsdatascience.com/lidar-3d-object-detection-methods-f34cf3227aea