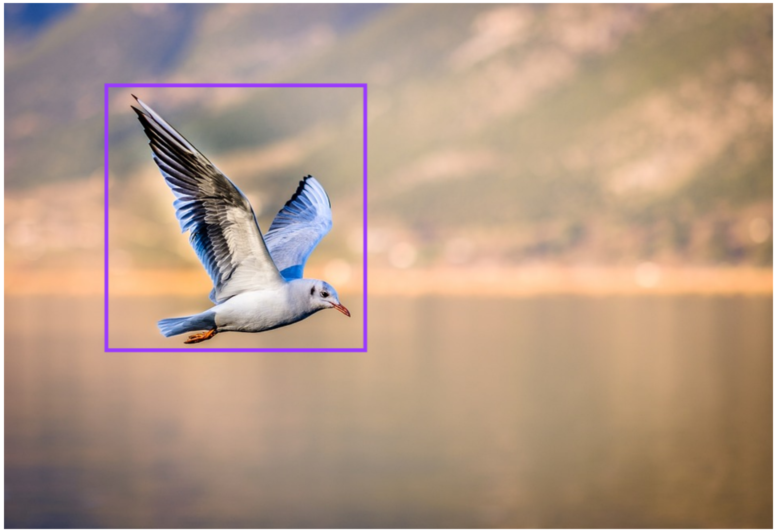
# Object detection

IN5400, spring 2021

- **Single-instance detection and localization** ← **Know enough to implement it**
- **Object detection (multiple objects)** ← **Know core principles and techniques**
- **Performance metrics** ← **Know the most common ones**

# Image classification and object localization

- *Classify* an image with (at most) *a single* object
- Draw a *bounding box* around the object

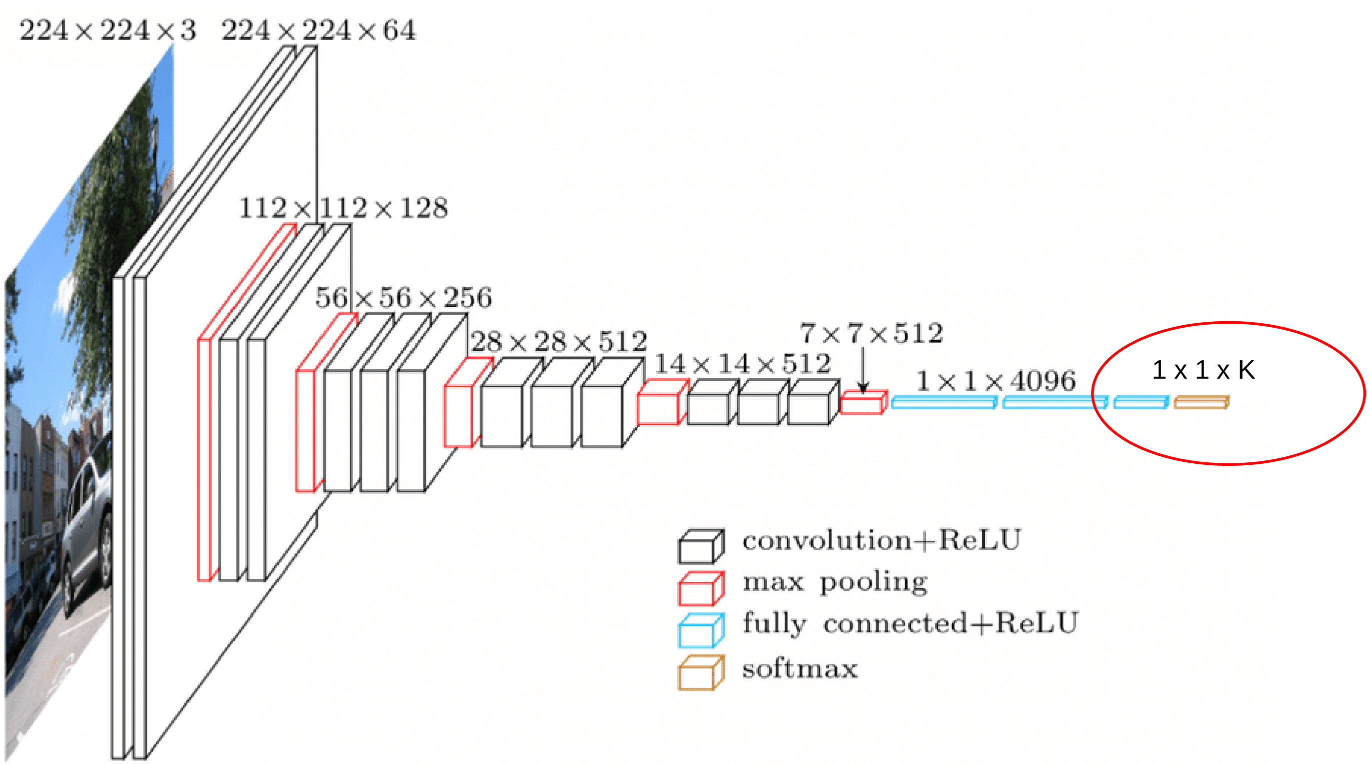


$b_r$: Center row coordinate
$b_c$: Center column coordinate
$b_h$: Box height
$b_w$: Box width

You already know this!

# Image classification

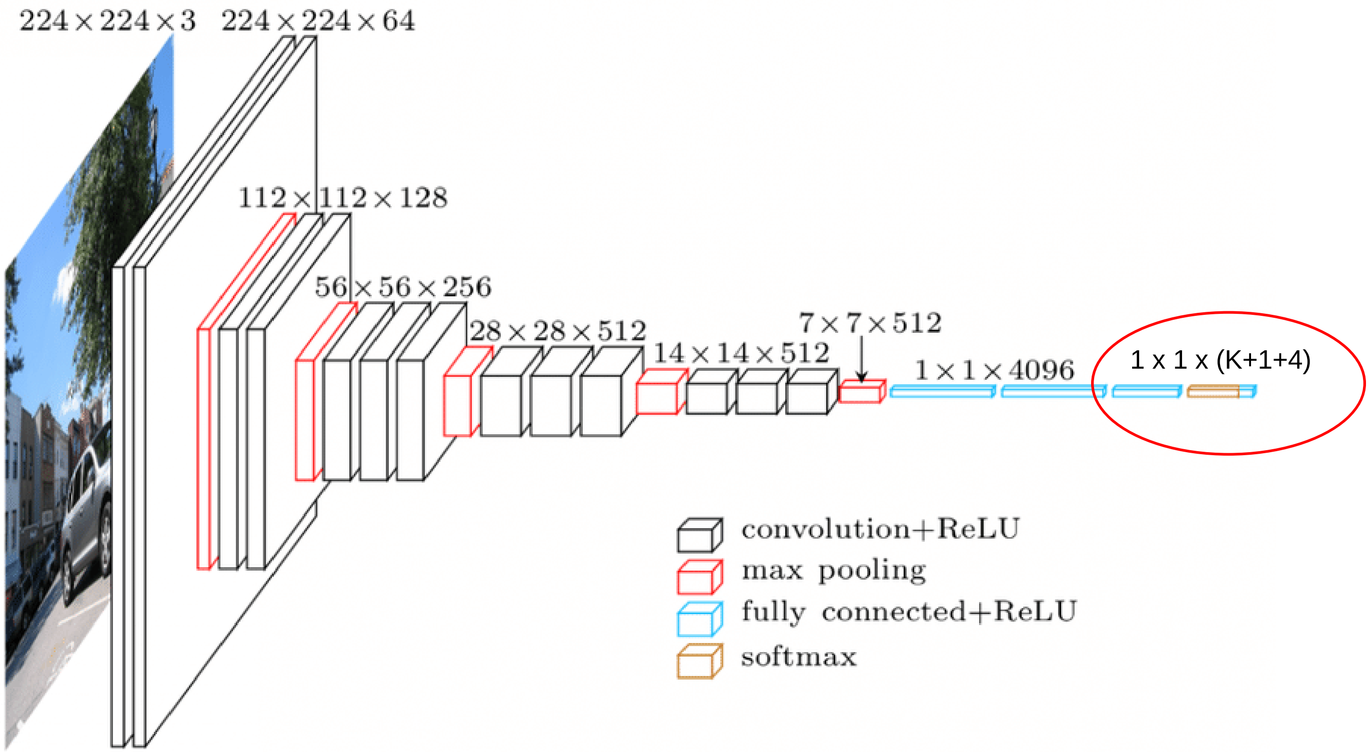
Network output




$$\mathbf{y} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_K \end{bmatrix}$$

The VGG16 network; however, think of it as just a conceptual illustration of a convolutional network

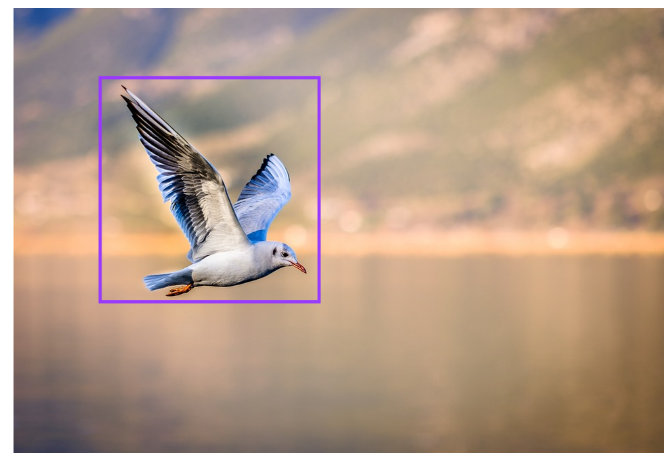
# Image classification + localization




$$\mathbf{y} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_K \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix}$$

Background class / no object

$$\mathbf{y} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_K \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array}\right\} \text{Class probabilities} \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{Bounding box} \end{array}$$



$b_r$: Center row coordinate
$b_c$: Center column coordinate
$b_h$: Box height
$b_w$: Box width

# Example I/III

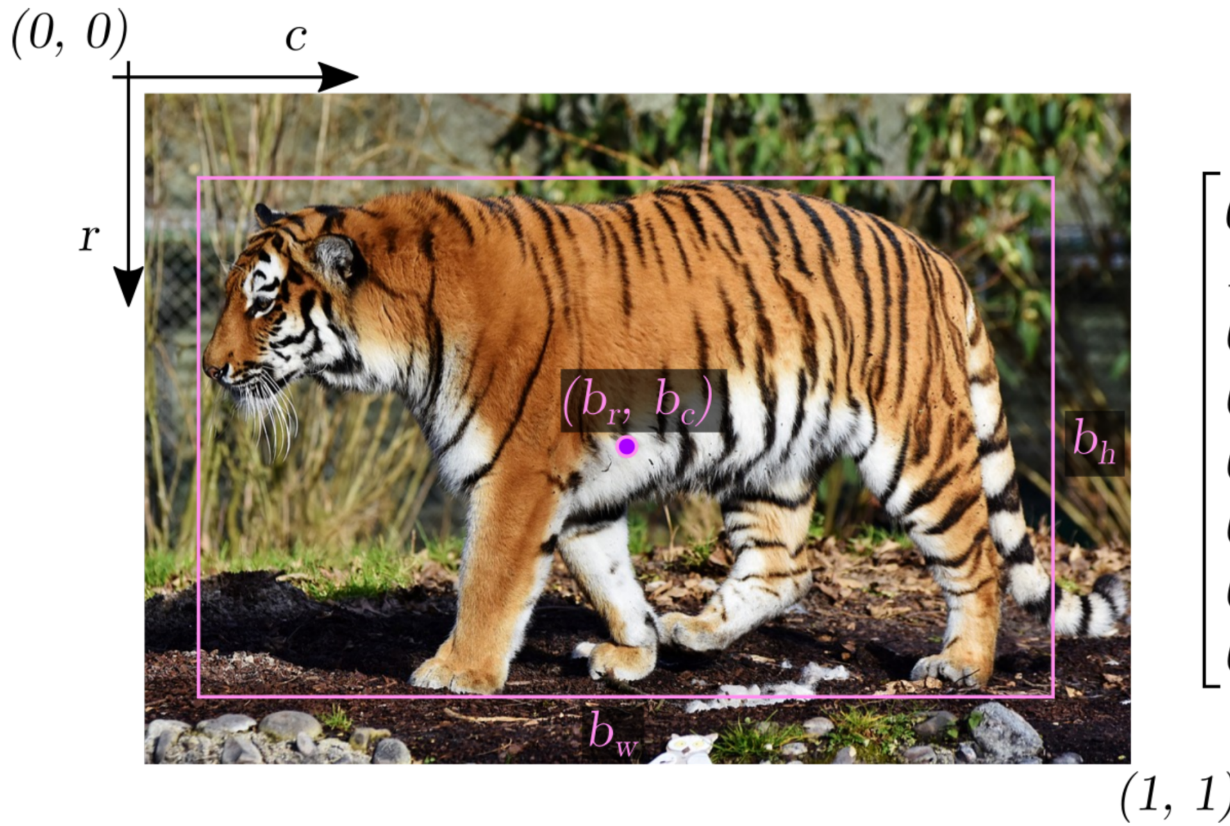
Note: These examples show what our network ideally should have as output for such images, cf. training data.

$c_1$: Tiger

$c_2$: Leopard

$c_3$: Lion

$$y = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix}$$




$$\begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.55 \\ 0.45 \\ 0.80 \\ 0.90 \end{bmatrix}$$

**Figure 17:** Tiger. Image source: https://www.pixabay.com

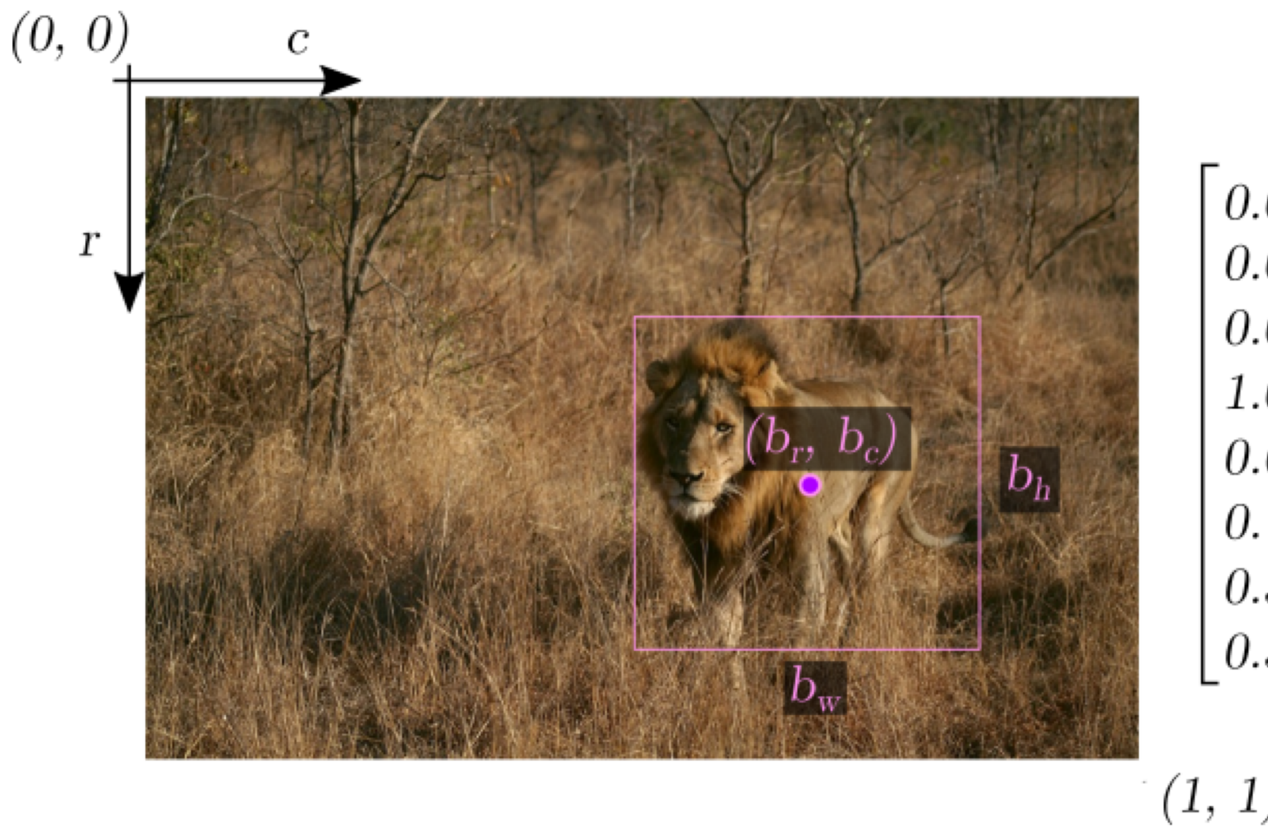
# Example II/III

$c_1$: Tiger

$c_2$: Leopard

$c_3$: Lion

$$y = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix}$$




$$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 1.0 \\ 0.65 \\ 0.70 \\ 0.35 \\ 0.35 \end{bmatrix}$$

**Figure 18:** Lion. Image source: https://www.pixabay.com

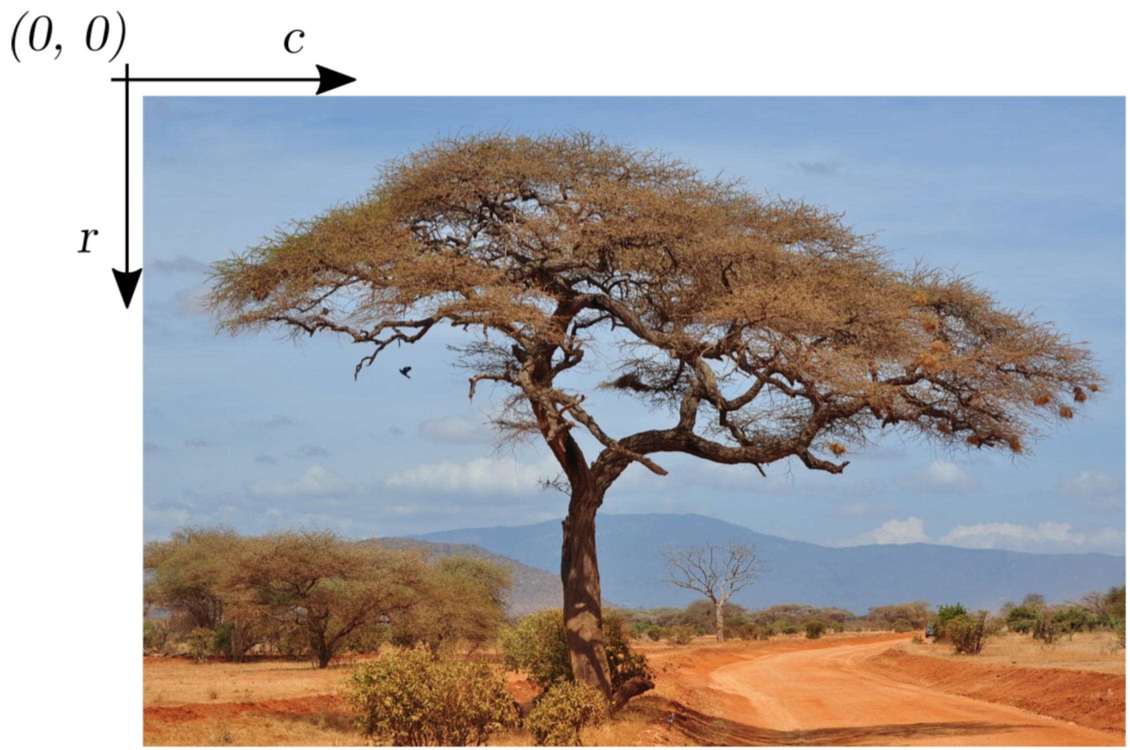
# Example III/III

$c_1$: Tiger

$c_2$: Leopard

$c_3$: Lion

$$y = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix}$$




$$\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \varnothing \\ \varnothing \\ \varnothing \\ \varnothing \end{bmatrix}$$

Figure 19: Savannah. Image source: https://www.pixabay.com

Ø : We do not care!

# Loss function

- Partition $y$ into $y = [c, b]$, with
  - $c = [c_0, c_1, \ldots, c_{N_c}]$
  - $b = [b_r, b_c, b_h, b_w]$

- $L_2$ loss for object bounding box location $b$

$$L_b(b, \hat{b}) = \sum_{i \in \{x,y,h,w\}} (b_i - \hat{b}_i)^2$$

L1 or "smooth-L1" also common

- Cross entropy loss for object categories $c$

$$L_c(c, \hat{c}) = -\sum_{i=0}^{n} c_i \log \hat{c}_i$$

- The total loss can be written as

$$\boxed{L(y, \hat{y}) = L_c + [c_0 = 0] L_b}$$

The terms can also be weighted

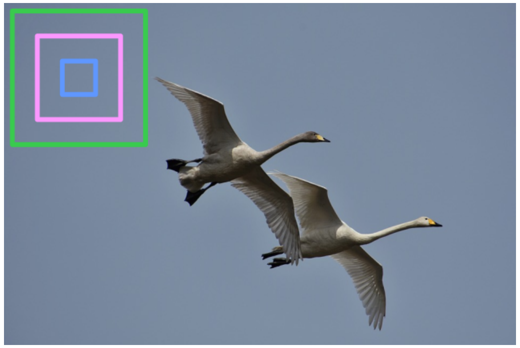
Only compare bounding box if there is an object

# Multiple objects

- (Sliding windows)
- Single stage networks
- Region proposal algorithms

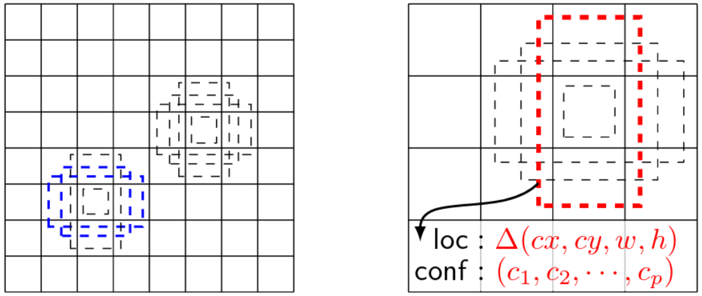
# (Sliding window approach)

- Slide (multiple-sized) windows across the image
- Apply an image classifier on every location
- Extract local score-peaks
- OK for “cheap” classification methods
- Very slow for CNN classifiers



# Towards multiple object detections I/II

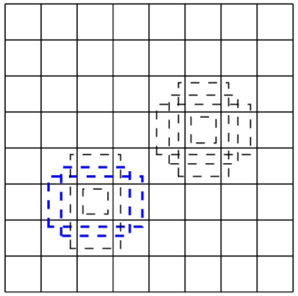
- **Simple idea:** Make the network output *many bounding boxes* (and class-belongings)!
- How to assign these outputs to the different objects?
  - Let them each have a *default box*, and let them find and predict objects that have a similar shape and location



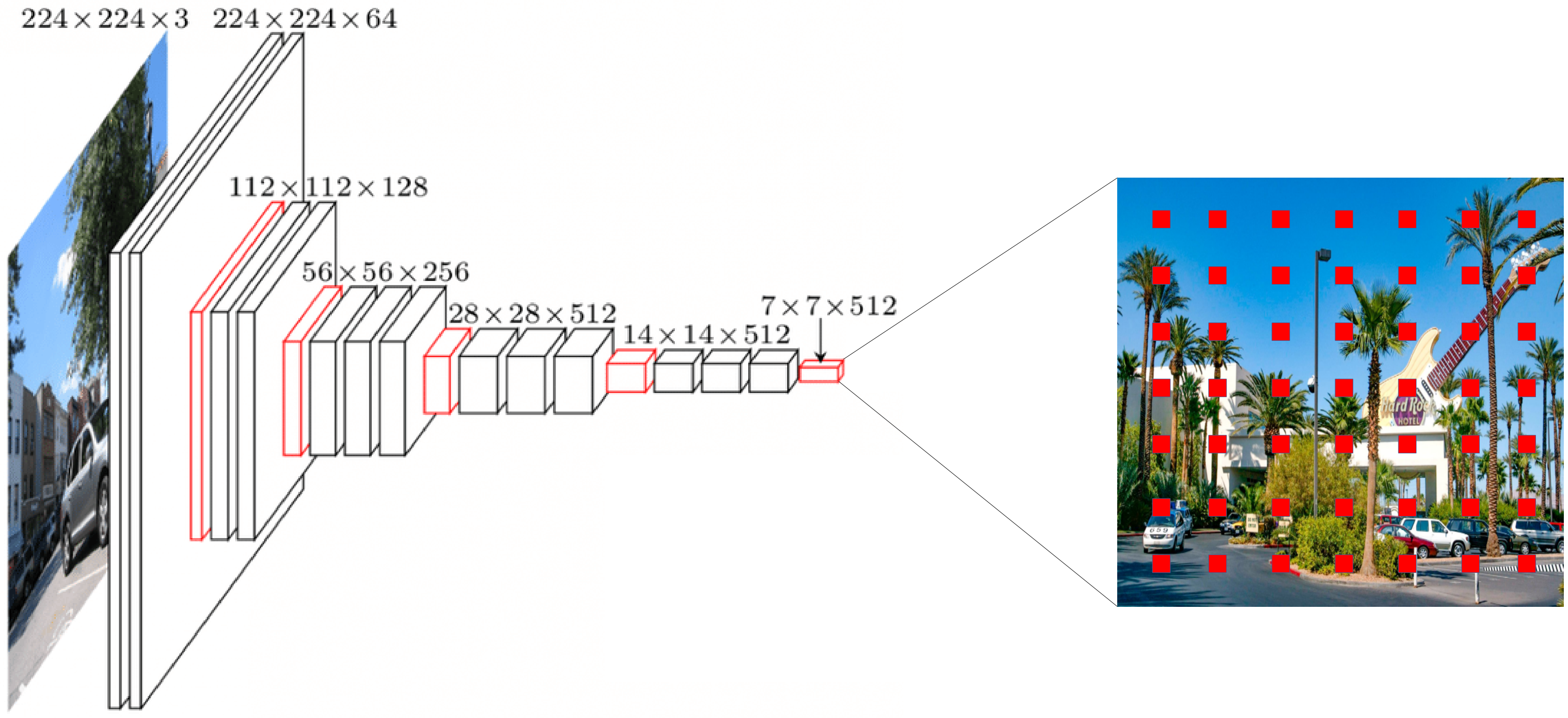

"Default boxes" also go by the name *anchor boxes* or *priors*.

# Towards multiple object detections II/II

- With *thousands* of predicted boxes, straight-forward implementation *too flexible* → impossible to train
- Network design and re-use of weights is important
- Key elements:
  - Let the identically-shaped boxes be predicted by a shared set of weights
  - Convolutions all the way to class+bbox predictions

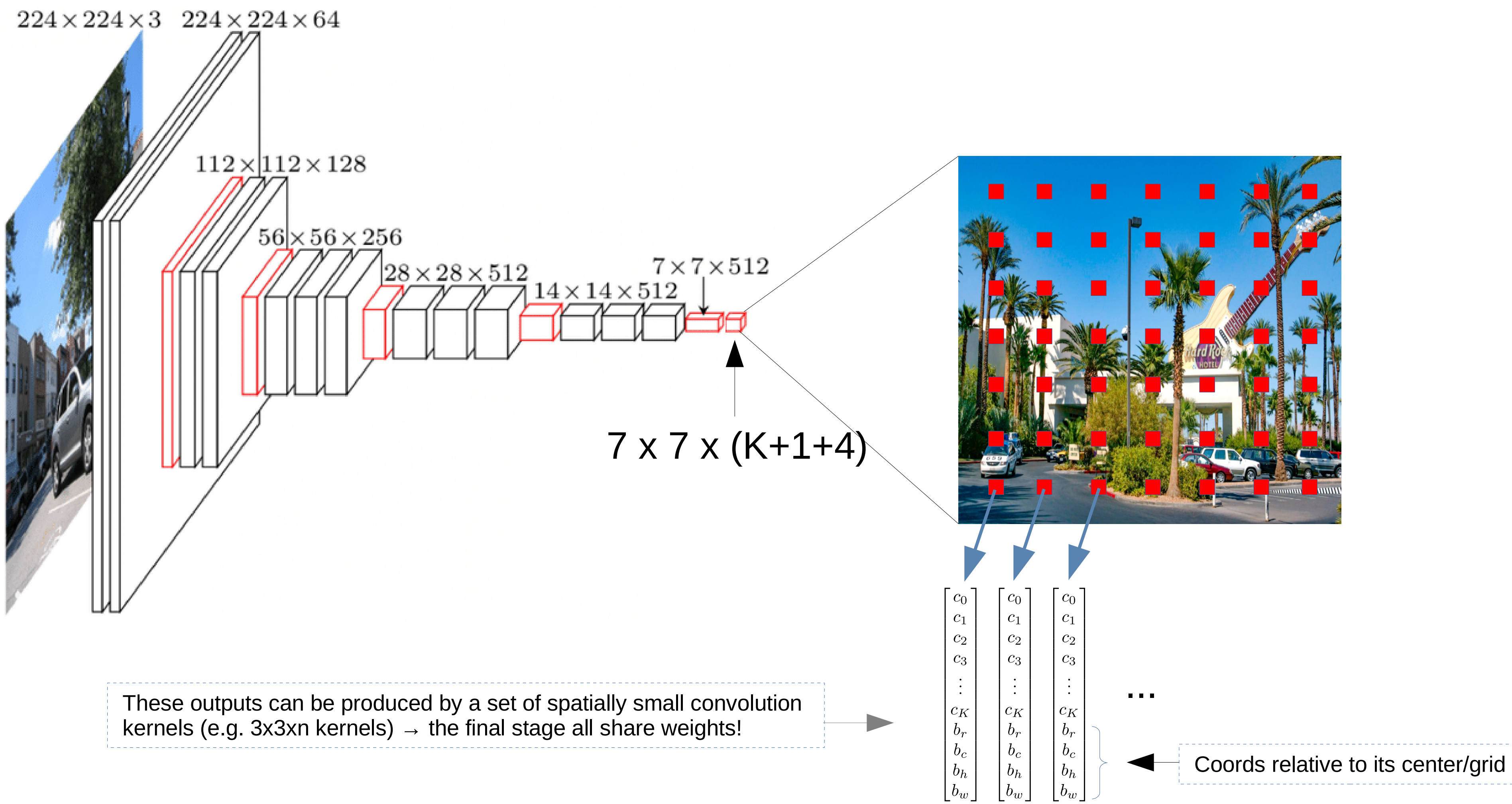


# Feature maps



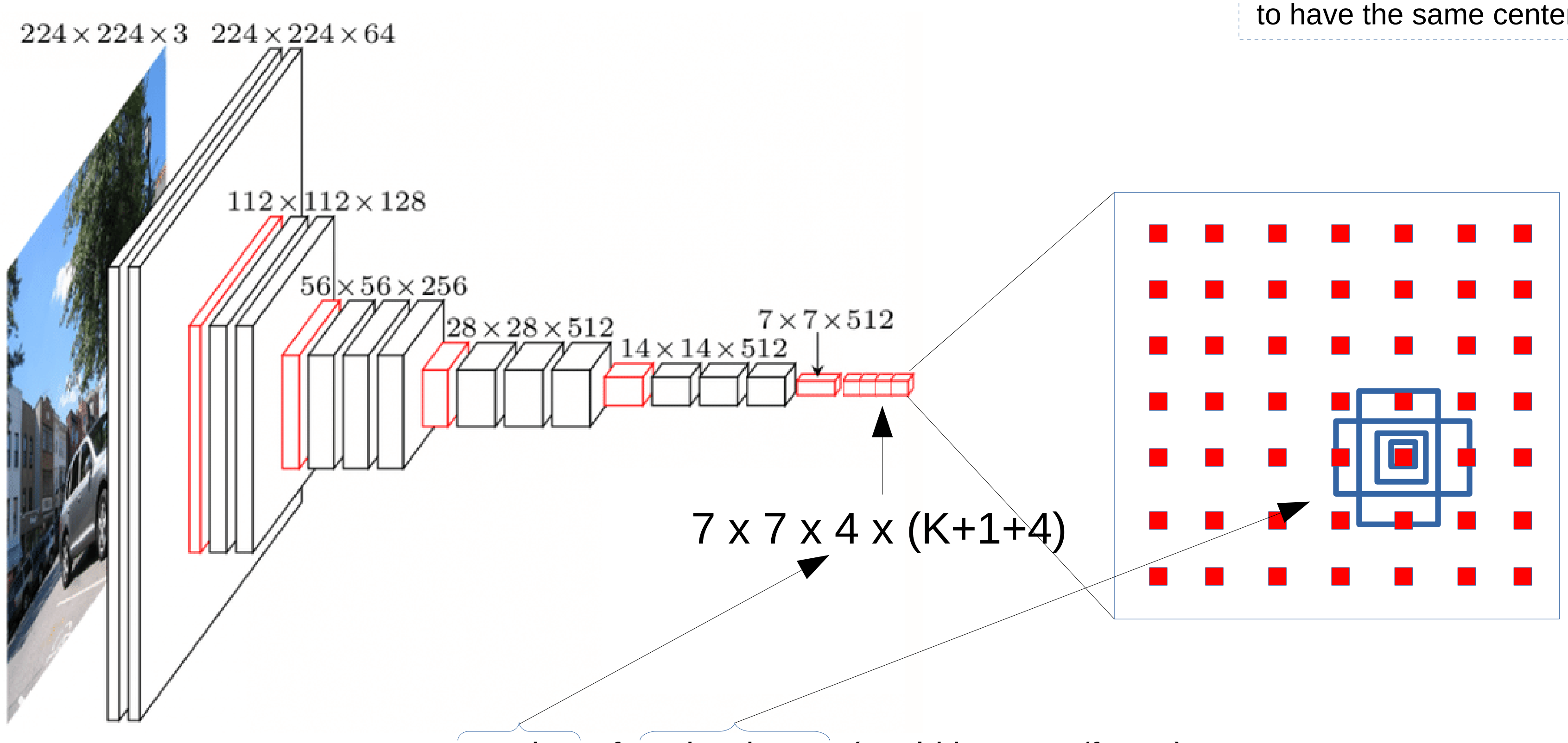

Your network already produces features that are spatially distributed

# Convolutions all the way



# .. more predictions per location

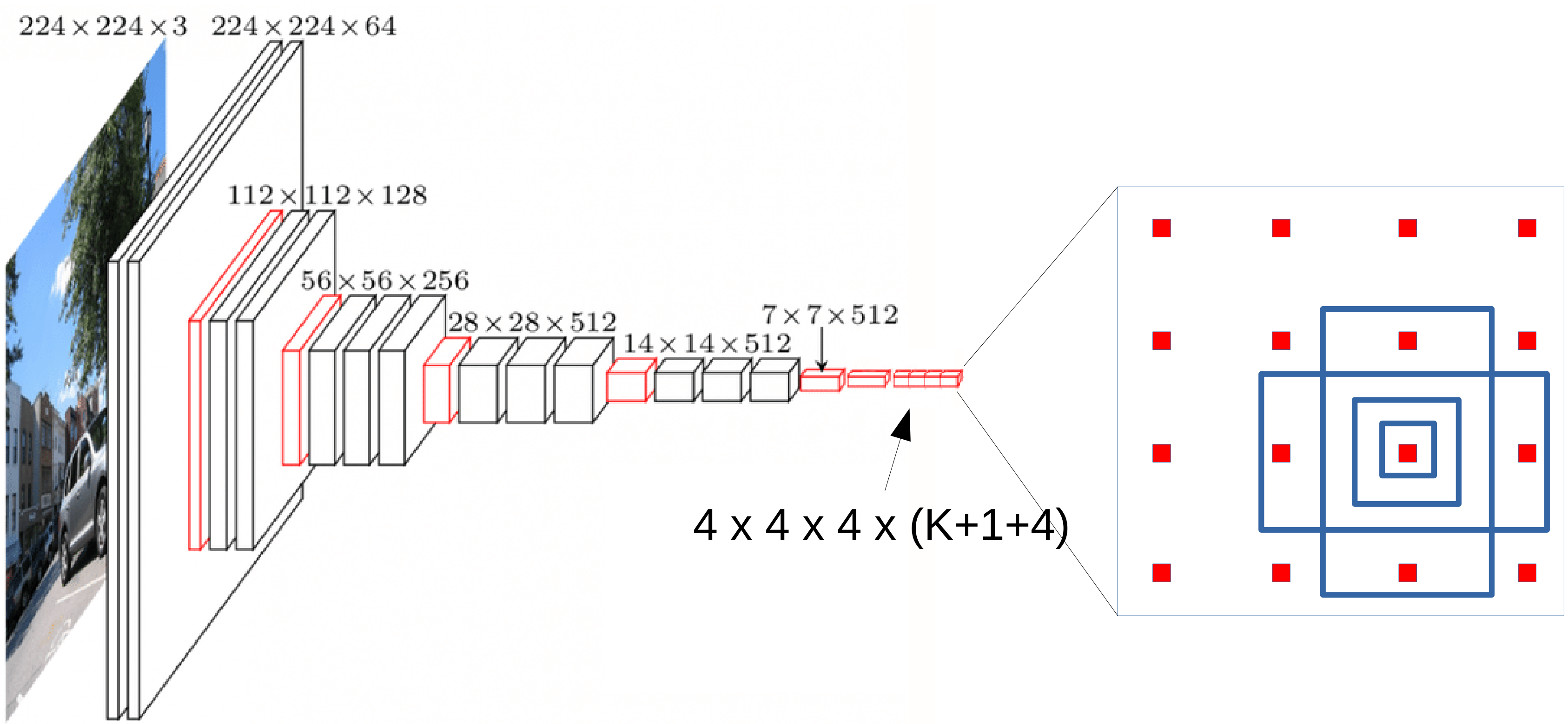
Anchor boxes allow multiple objects to have the same center location




Number of anchor boxes (could be more/fewer)

Sometimes anchor boxes go by the name *priors* or *default boxes*.

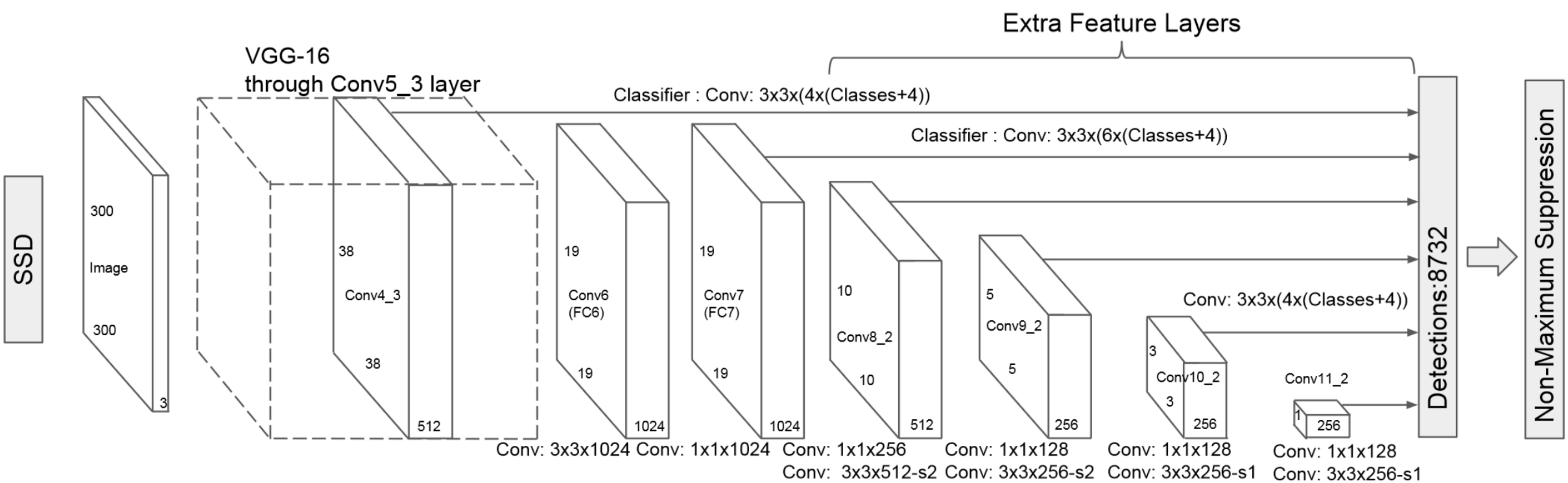
# Subsampling and box size




Likely *fewer* bigger boxes needed

.. and, if still using 3x3 convolutions, the bigger boxes is predicted using a larger image context

# Single-shot multibox detector (SSD)




Liu, Wei, et al. "SSD: Single shot multibox detector", ECCV, 2016.

Even though this “simple” design works well in practice, we visit some common-seen refinements/improvements later.

Note that the YOLO algorithm, RetinaNet and SSD have strong similarities and are often grouped together (“single-stage detectors”).

# Non-max suppression I/III

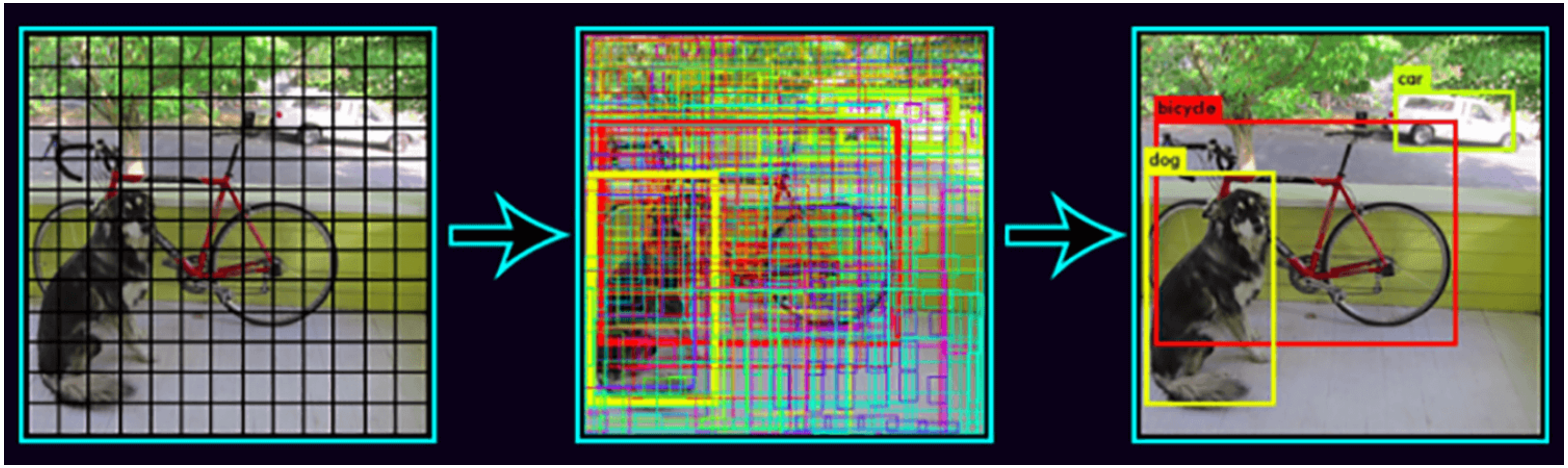
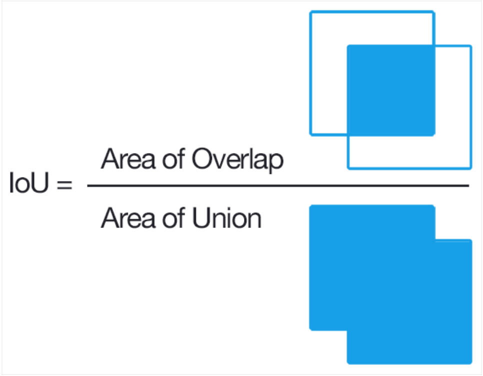
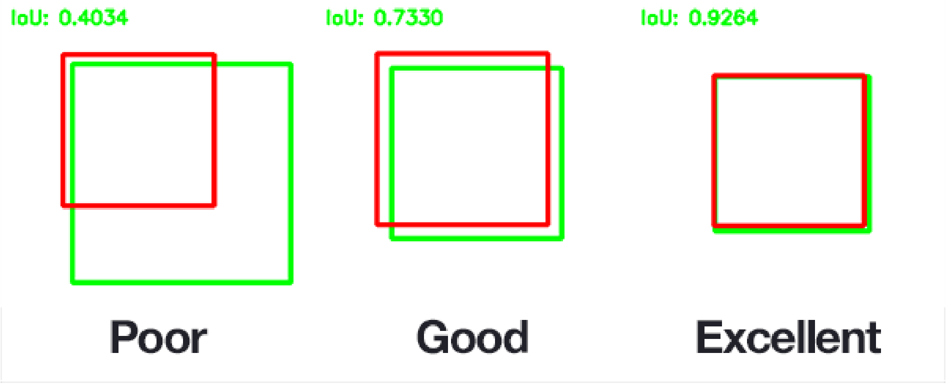



Illustration from the YOLO 2015 paper

# Non-max suppression II/III

Intersection over Union (IoU)




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

# Non-max suppression III/III

- Important step in several object detection algorithms
- Remove all boxes having no $c_1, c_2, .., c_K$ larger than, say 0.5

- For each class $i = 1, 2, \ldots, n$
  - Create a list of "unseen" regions $U_i$ that contains all the regions in the image
  - Create an empty list of regions to keep $K_i$
  - While there are regions left in $U_i$
    - Find the most probable region $R_{\max}$
    - $R_{\max}$ can be the region with highest value of $c_i$ (or some similar criterion)
    - Remove all regions that overlaps with $R_{\max}$ (e.g. with $iou > 0.5$), from $U_i$
    - Move $R_{\max}$ from $U_i$ to $K_i$

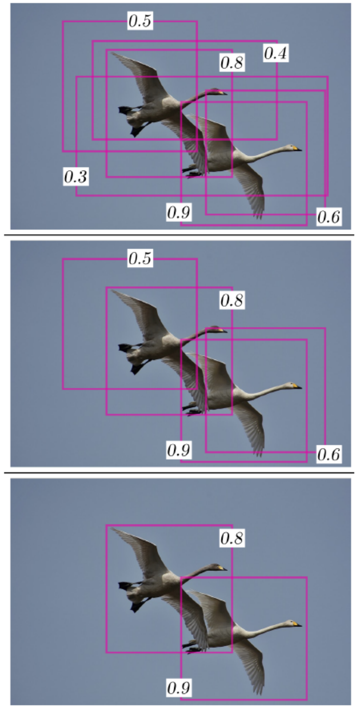



**Figure 29:** Top: Original. Middle: Too low $c_0$ removed. Bottom: $iou > 0.5$ removed. Image source: https://www.pixabay.com

# Notes on how to train I/III

- Match anchors with ground truth boxes:

```
for every ground-truth box:
    match the ground-truth box with anchor-box having the biggest IoU

for every anchor-box:
    ious = IoU(anchor-box, ground_truth_boxes)
    max_iou = max(ious)
    if max_iou > threshold:
        i = argmax(ious)
        match the anchor-box with ground_truth_boxes[i]
```

That is, how to go from human annotated ground truth to “ideal” network outputs

# Notes on how to train II/III

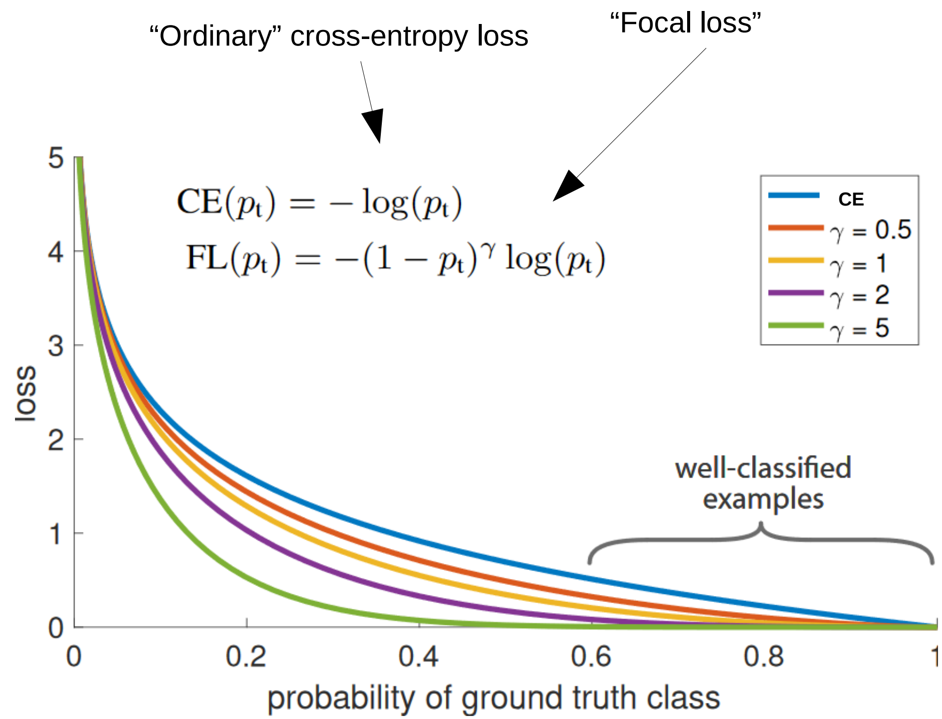
- Often many more anchor-boxes without a match → *imbalanced dataset*
  - Hard Negative Mining
    - Select only *the most difficult* background patches (lowest background score) when computing loss
  - Change loss function to downscale importance of highly certain background patches → "*focal loss*" (see later slide)

# Notes on how to train III/III

- Data augmentation (as always..)
  - Create more (and plausible so) data; cropping, resizing, mirroring, photometric distortions, ...

# Focal loss

- “Focal loss .. focuses training on a sparse set of hard examples”
- Vigor of this focus controlled by a γ-parameter
- One can in addition add more weights to the non-background patches
  - Cf. the α factor often mentioned in conjunction with focal loss




Let’s say we have many background patches which we are quite certain about being correctly labeled (p ≈ 0.7).

Using the CE-loss, we can substantially reduce this loss by going from “quite certain” to “very certain” (p ≈ 0.9), not so for the FL-loss.

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

Lin et al.. "Focal Loss for Dense Object Detection", ICCV, 2017.

# Near-infinite set of refinements..

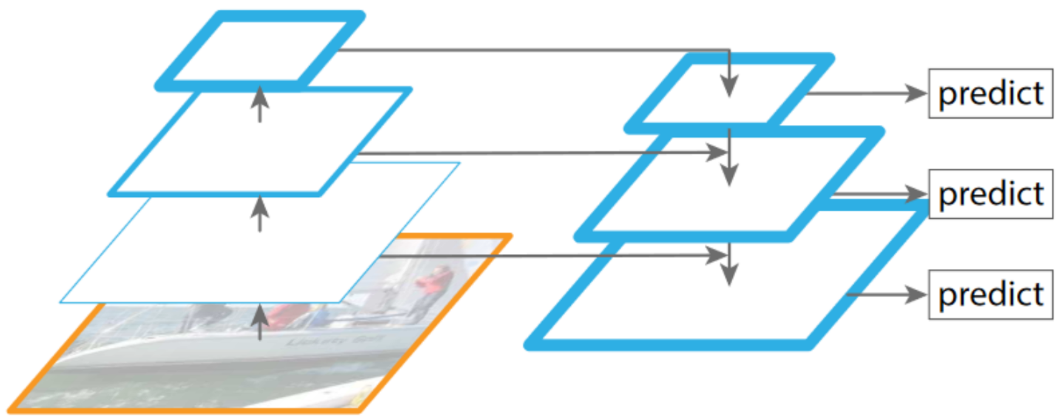
- "Backbones"
  - VGGs / ResNets / ResNeXts / ResNeSts / ..
- Pretraining
  - What data and tasks are they trained on?
- Methods/implementations
  - SSD / RetinaNet / YOLO / YOLOv2 / YOLOv3 / ..
- Many versions and combinations of concepts (adaptive anchor-boxes, layer-merges, losses..)
- However, the ubiquitous "FPN" needs special attention ... (next slide)

# Feature-pyramid networks (FPN) I/II

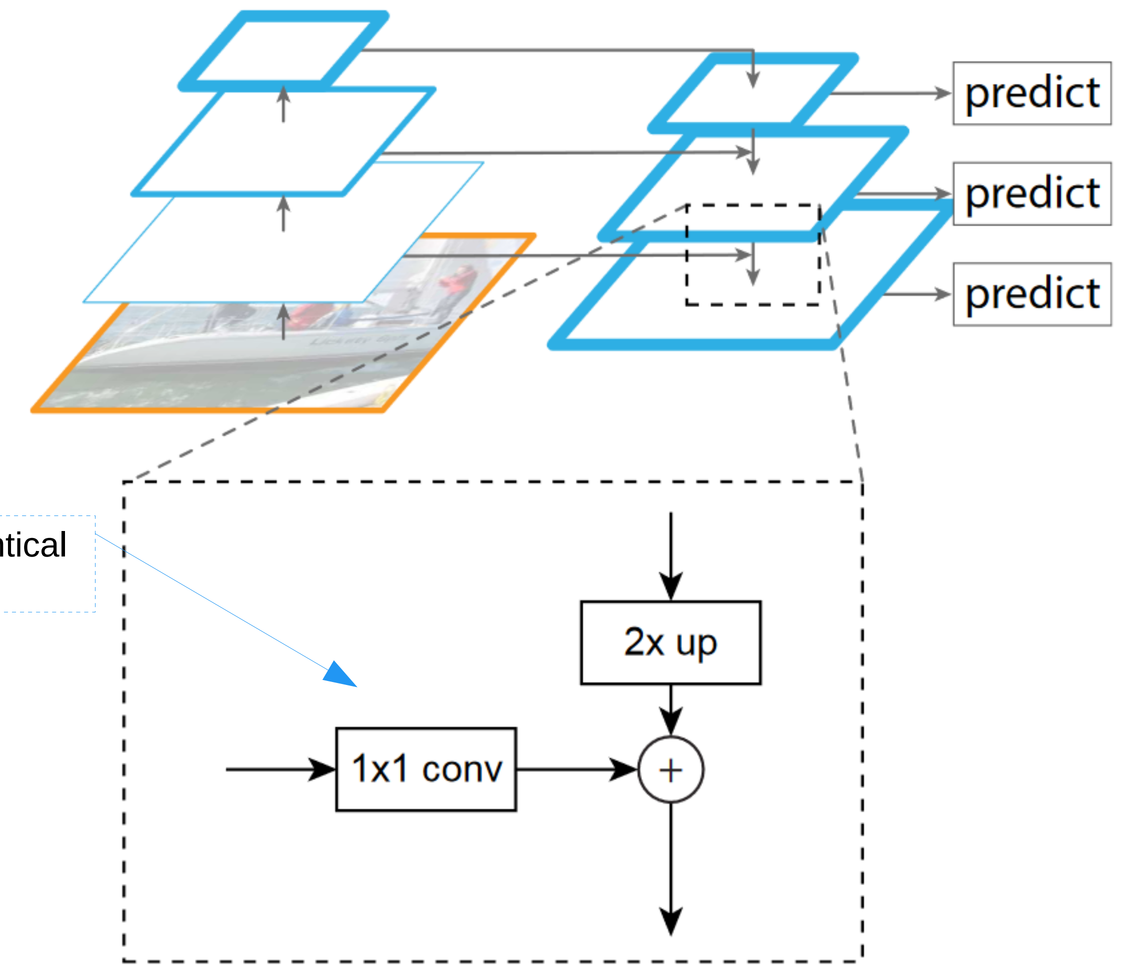
- Cf. the SSD illustration (slide 18)
- Deeper layers → semantically stronger features
  - Only the largest anchor boxes benefit from this
- Let us try to propagate some of this “semantic strength” into the earlier layers before detecting our objects!

Note that there are many different ways of achieving this, however the ”FPN” described on the next slides is easily implemented, fast and does wonders in practice..

# Feature-pyramid networks (FPN) II/II



# Feature-pyramid networks (FPN) II/II




Ensures all layers have identical dimensions (e.g. d = 256)

Lin et al. "Feature Pyramid Networks for Object Detection", CVPR, 2017.

# Region proposal algorithms

- A subclass of object detection methods
- Use a separate method to find candidate regions
- Filter out regions without an object, or redundant, overlapping regions with an object
- Classify these regions and refine region boundary

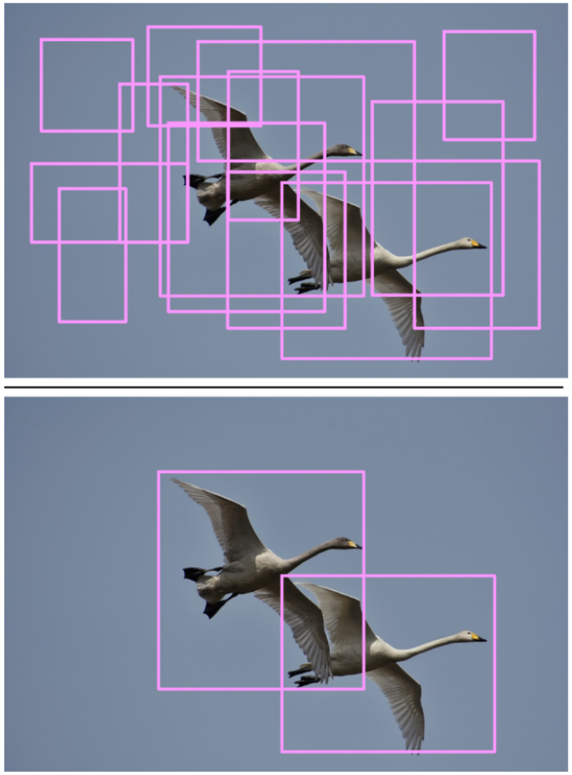


Figure 23: Image source: https://www.pixabay.com

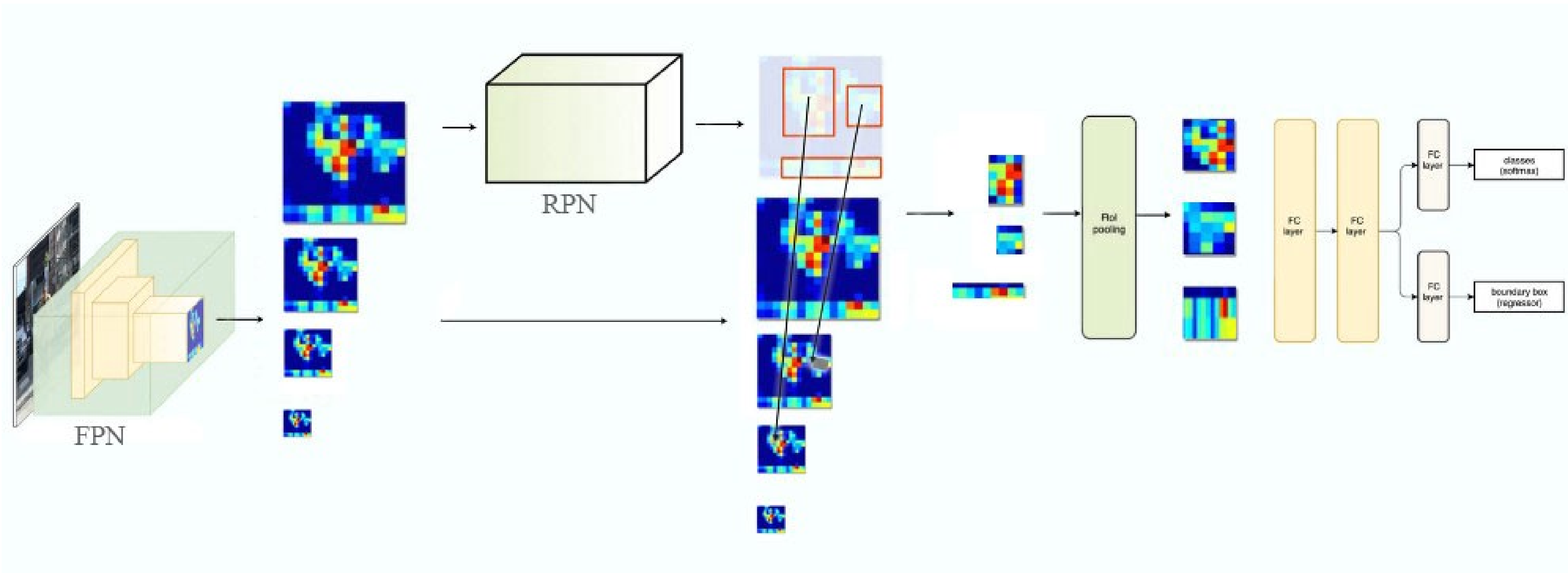
# Example: Faster R-CNN I/II

- **Stage 1:** Region proposal network (RPN) similar to a simplified SSD/RetinaNet (background / no background only)

➢ Note that we again have some “backbone” and e.g. a FPN step

- **Stage 2:** Regions (at the *feature level*, not pixel level) are resampled to a *fixed-size patch*, and fed into a refinement network which classifies and tunes the bounding boxes

# Example: Faster R-CNN II/II



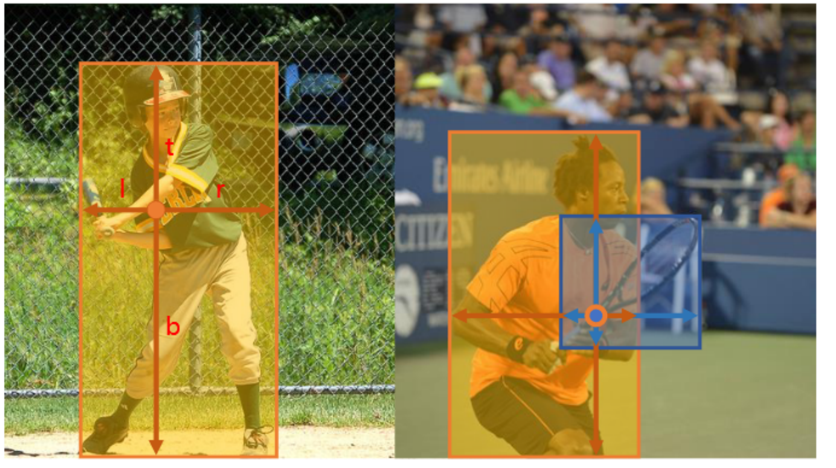

TODO: Source of illustration.

- Traditionally more precise than the one-stage detectors, however slower

Cursory syllabus

# “Anchor-free” approaches

- No pre-defined set of anchor boxes
- Example: FCOS
- Instead of anchor boxes, we output:
  - class
  - (left, right, top ,bottom)
  - “centerness”




Tian et al, “FCOS: A simple and strong anchor-free object detector", 2020.

# Performance and evaluation metrics

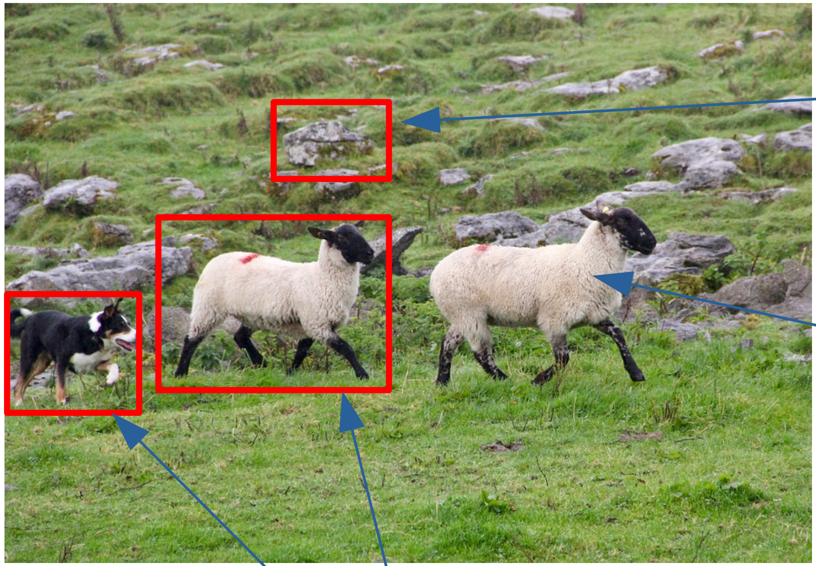
- Precision
- Recall
- Average precision (AP) / mean average precision (mAP)

Per class!

.. over classes

*Single number* covering all classes!

Assuming a ***single class***: “animal”



FP (false positive)

FN (false negative)

TP (true positives)

Note: Requires an ***IoU threshold***, e.g. 0.5 or 0.75.

Remember: Precision/recall is calculated **per class**!

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

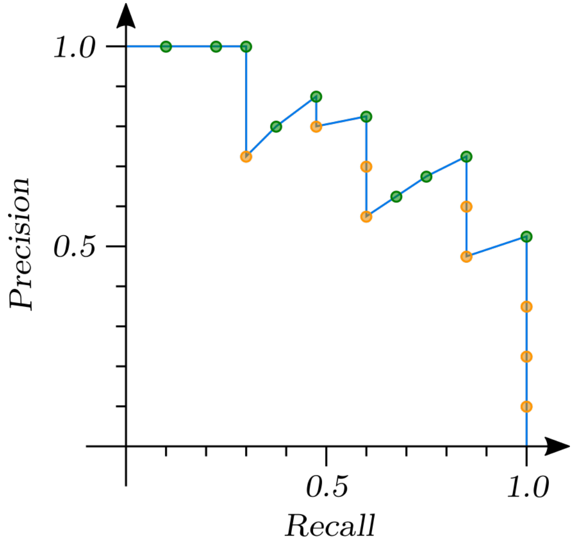
(How many of our predictions are actually true/objects)

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

(How many of the objects of interest are found)

# Precision-recall curves

- All detected objects have a class score, $c_k$, for each class k
- If we alter the score-threshold for what constitutes a predicted object of a given class, the precision and recall for this class will change
- Typically, lowering the threshold gives more detections, which increases the recall but lowers the precision
- By *changing this score-threshold,* we get a ***curve*** for each class

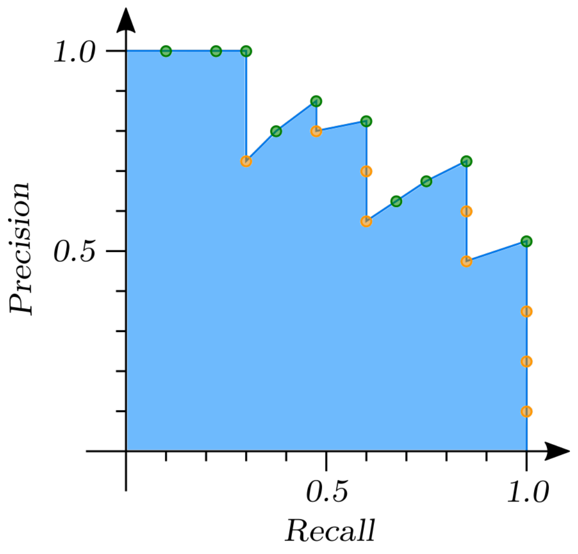



Remember that such curves depend on an ***IoU threshold.***

# Average precision (AP)

- For a given class, the average precision is the **area** under its precision-recall curve (or some *approximation* of it)

- Note, we have an $AP_k$ for each class k



# Mean average precision (mAP)

- The mean average precision (mAP) is simply the average AP over all the classes:

$$\mathrm{mAP} = \frac{1}{K}\sum_{i=1}^{K} \mathrm{AP}_i$$

One can also average over different IoU thresholds, e.g. as in the COCO and PASCAL VOC challenges.

# In practice

- PyTorch/torchvision
  - Currently RetinaNet and Faster-RCNN
- pytorch.org/hub/
- MMDetection, Detectron2
  - Actively developed toolboxes
  - A wealth of “backbones”, architectures and techniques

# Summary

- Single-instance detection and localization
  - Add network outputs, provide examples (training data), specify loss

- Object detection (multiple objects)
  - Single-stage networks | SSD/YOLO/RetinaNet
    - Many bounding boxes + classifiers
    - Default boxes / priors / anchor boxes
    - Re-use of weights → convolutions all the way..
  - Semantic richness on all feature levels → e.g. FPN
  - Region-proposal approaches | Faster R-CNN

- Performance evaluation metrics | Precision, recall, IoU, AP, mAP