UNIVERSITY
OF OSLO

IN4310/3310 Object Detection

dhananjt@ifi.uio.no



Pinterest's use of object detection

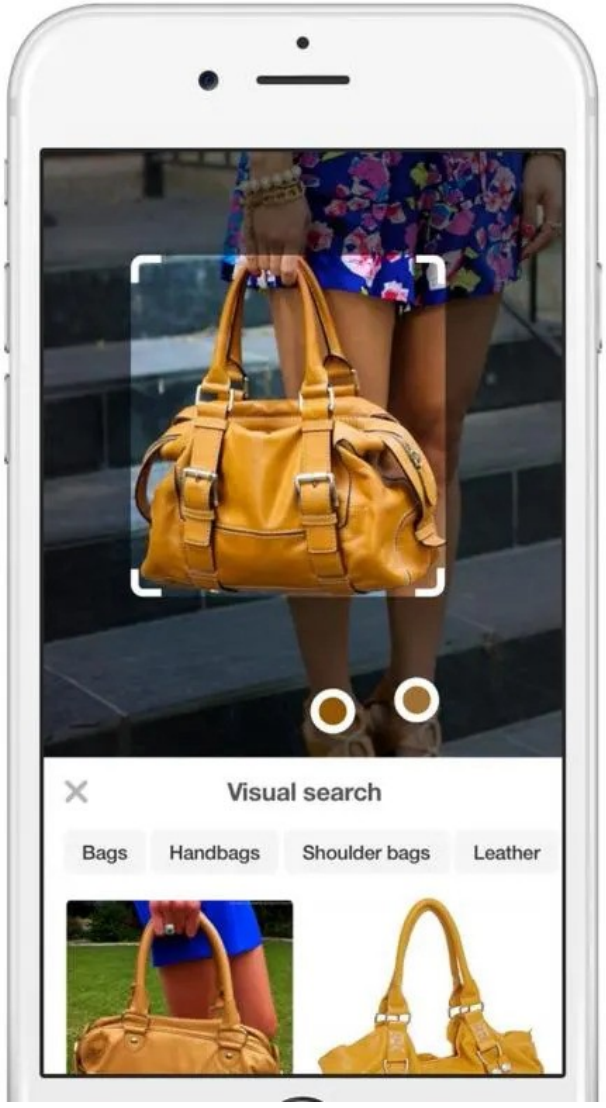


Figure 1: Image source: Pinterest's blog post on medium

Motivation: Google Lens

https://www.youtube.com/watch?v=BH_6BdgTdiw

Outline

1. Single-instance detection and localization
2. Object detection (multiple objects)
   - 2.1 Two stage detectors and one stage detectors
   - 2.2 A popular two-stage detector: Faster R-CNN
   - 2.3 Generating candidates using anchors
   - 2.4 Problem 1: Different object sizes
   - 2.5 Problem 2: Multiple overlapping predictions
   - 2.6 Notes on how to train
   - 2.7 Problem 3: Too many background predictions
3. DEtection TRansformer (DETR) for object detection
4. Performance and evaluation metrics

Outline

1. Single-instance detection and localization
2. Object detection (multiple objects)
   - 2.1 Two stage detectors and one stage detectors
   - 2.2 A popular two-stage detector: Faster R-CNN
   - 2.3 Generating candidates using anchors
   - 2.4 Problem 1: Different object sizes
   - 2.5 Problem 2: Multiple overlapping predictions
   - 2.6 Notes on how to train
   - 2.7 Problem 3: Too many background predictions
3. DEtection TRansformer (DETR) for object detection
4. Performance and evaluation metrics

Image classification and object localization

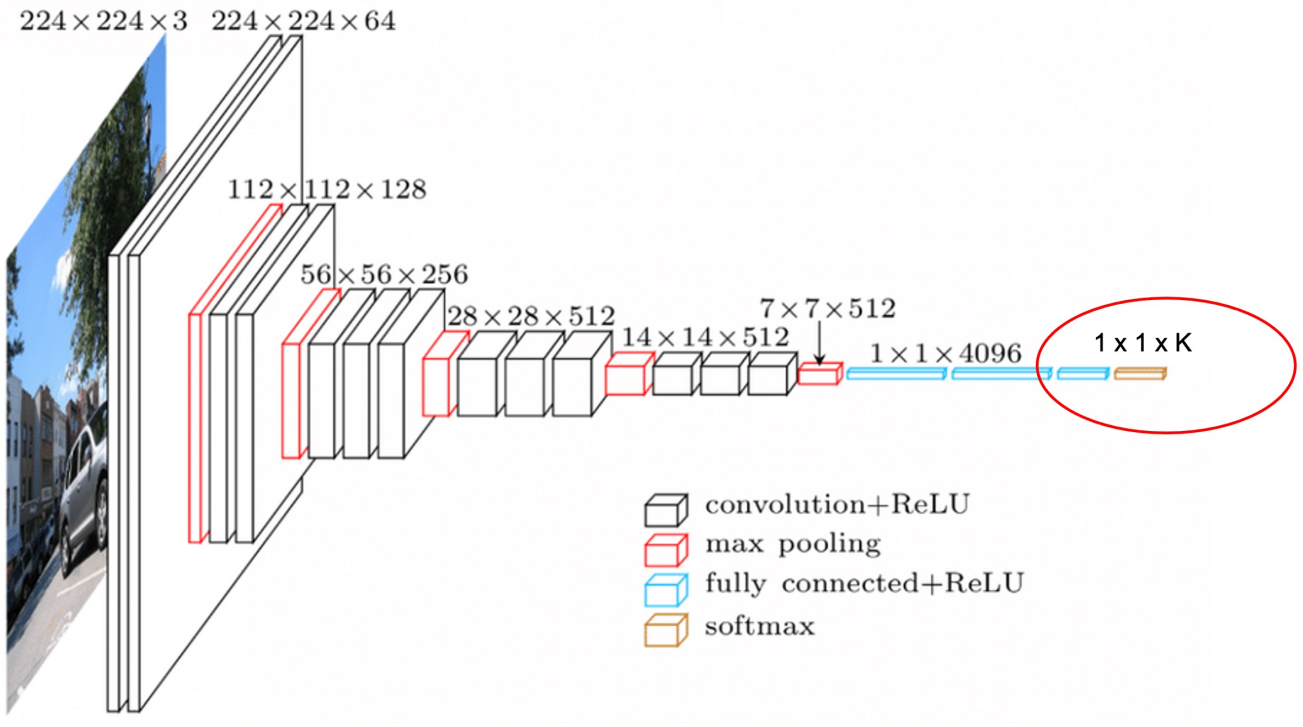
- *Classify* an image with (at most) *a single* object
- Draw a *bounding box* around the object



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

Image classification

You already know this!

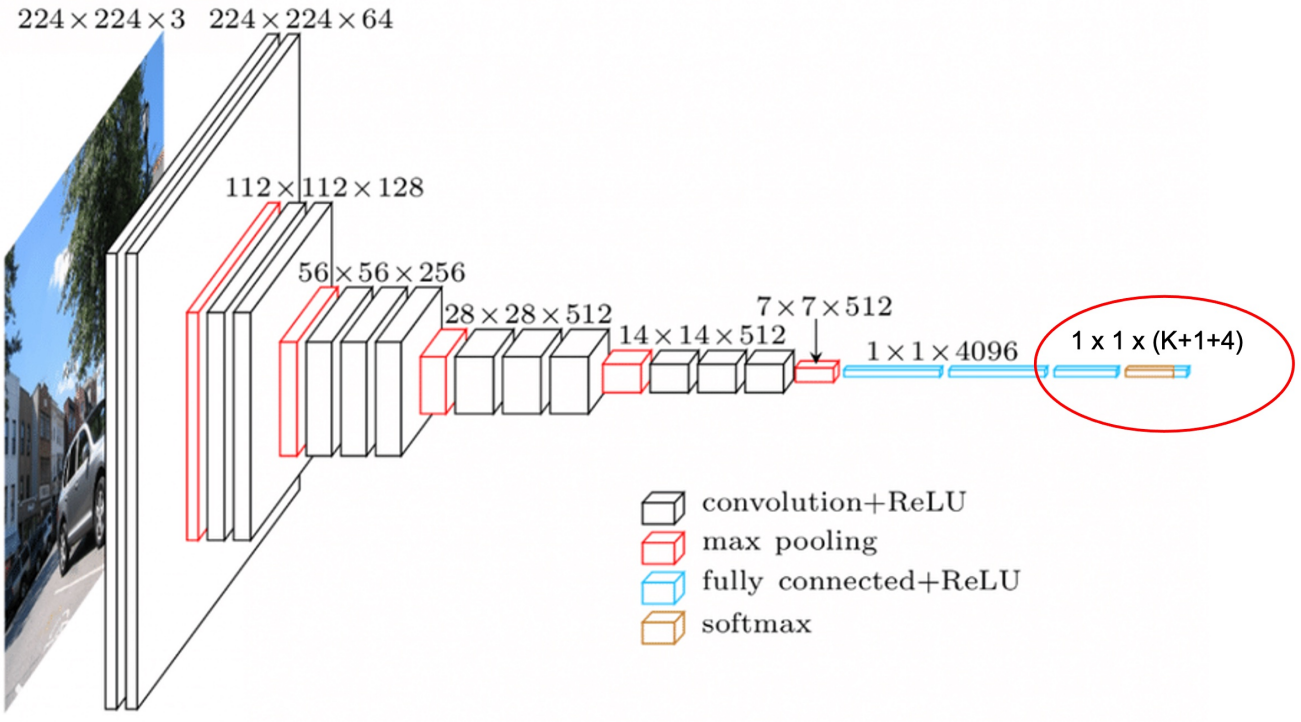


Network output

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

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

Image classification + localization



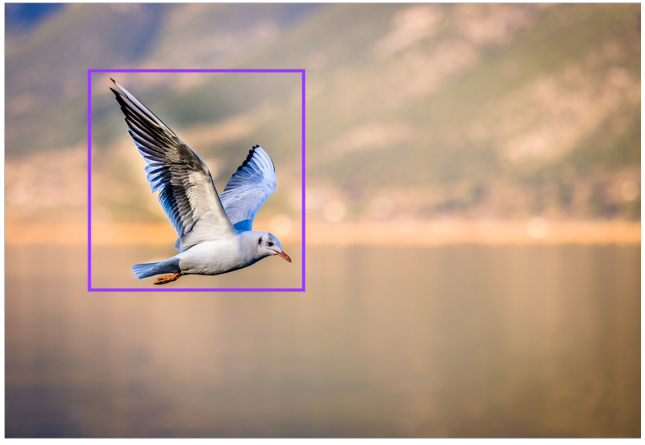
$$\mathbf{y} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_K \\ b_r \\ b_c \\ b_h \\ b_w \end{bmatrix}$$

Background class / no object

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

Class probabilities

Bounding box



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

Example I/III

c_1: Tiger

c_2: Leopard

c_3: Lion

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

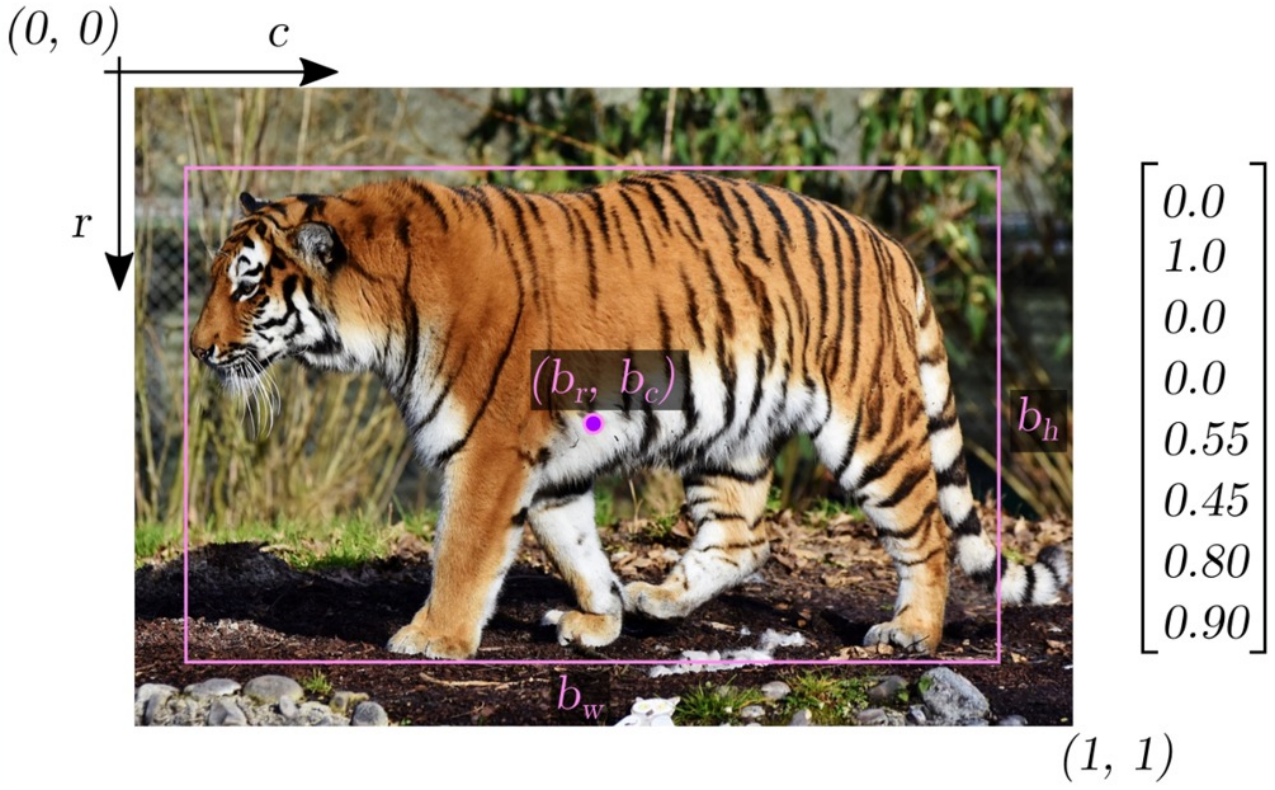


Figure 17: Tiger. Image source: https://www.pixabay.com

Example II/III

c_1: Tiger

c_2: Leopard

c_3: Lion

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

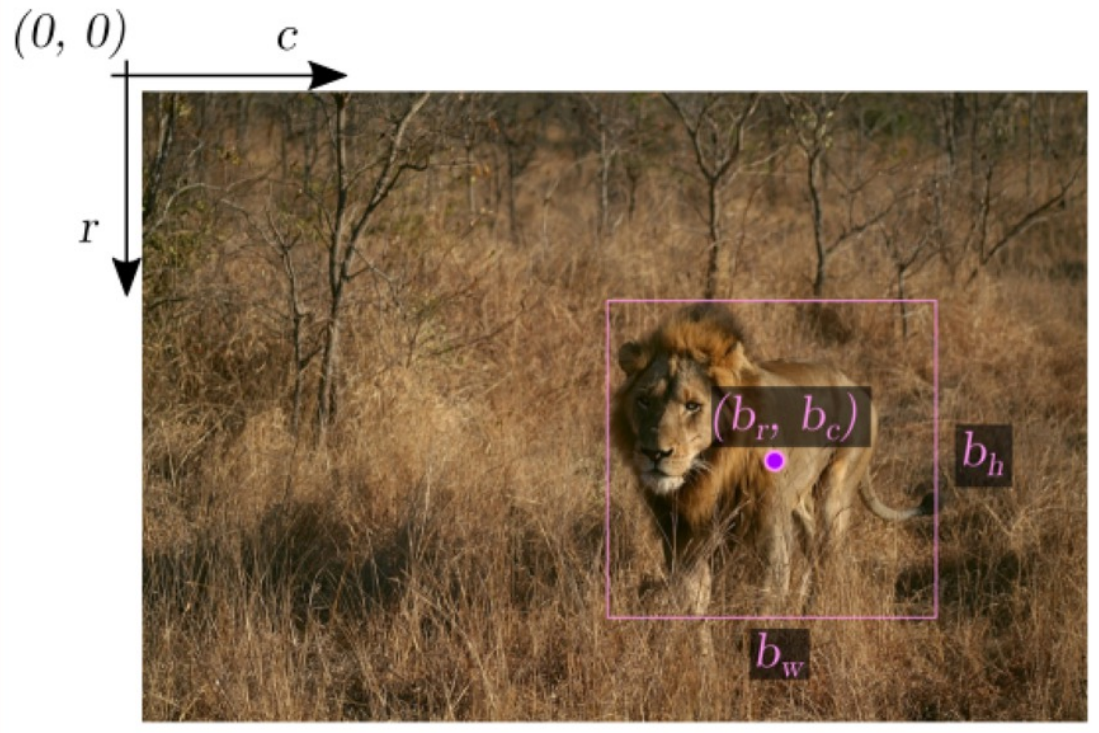


Figure 18: Lion. Image source: https://www.pixabay.com

Example III/III

c_1: Tiger

c_2: Leopard

c_3: Lion

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

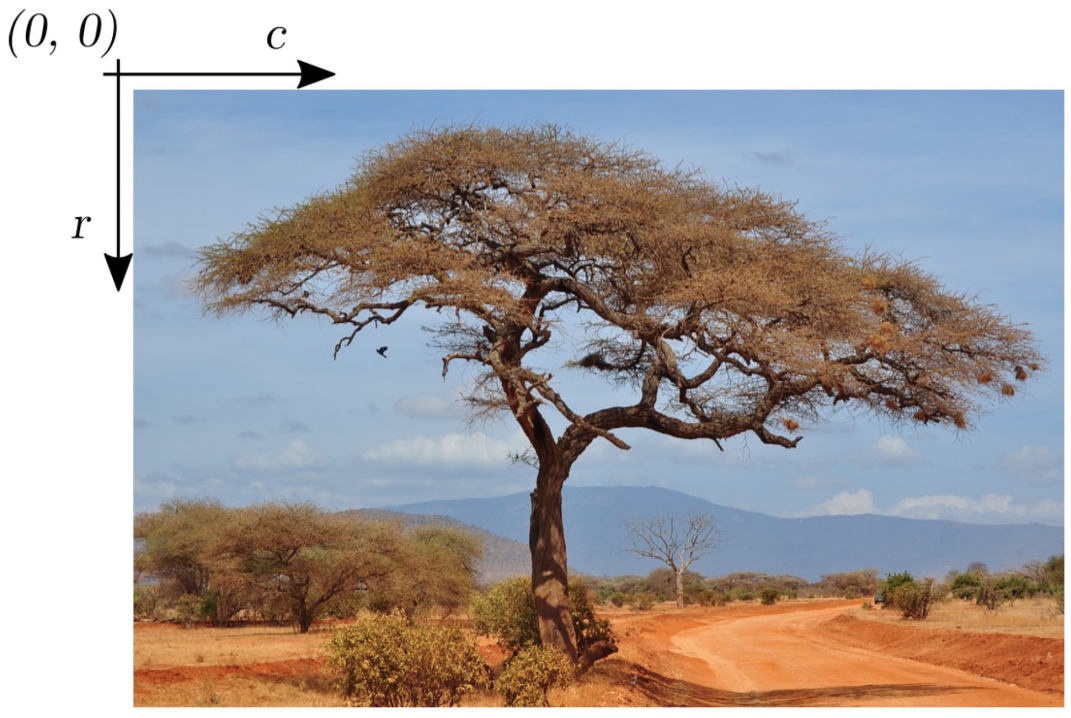


Figure 19: Savannah. Image source: https://www.pixabay.com

Loss function

- Partition y as y = [c, b] with
  - c = $[c_0, c_1, \ldots, c_N]$ representing scores for all categories
  - b = $[b_x, b_y, b_h, b_w]$ representing the bounding box
- Let c',b' be the ground truths.
- L_2 loss for the object's bounding box b

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

L1 or "smooth-L1" is also common

- Cross entropy loss for object categories c

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

- The total loss can be written as

$$L(y, y') = L_c + [c'_0 \neq 0] L_b$$

The terms can also be weighted

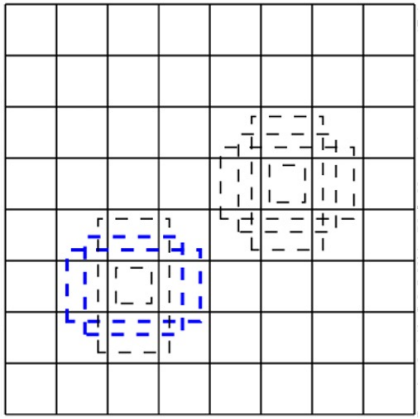
Compare bounding box only if there is an object

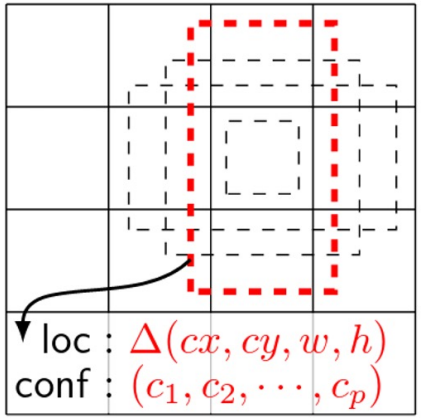
Outline

1. Single-instance detection and localization
2. Object detection (multiple objects)
   - 2.1 Two stage detectors and one stage detectors
   - 2.2 A popular two-stage detector: Faster R-CNN
   - 2.3 Generating candidates using anchors
   - 2.4 Problem 1: Different object sizes
   - 2.5 Problem 2: Multiple overlapping predictions
   - 2.6 Notes on how to train
   - 2.7 Problem 3: Too many background predictions
3. DEtection TRansformer (DETR) for object detection
4. Performance and evaluation metrics

Towards multiple object detections

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

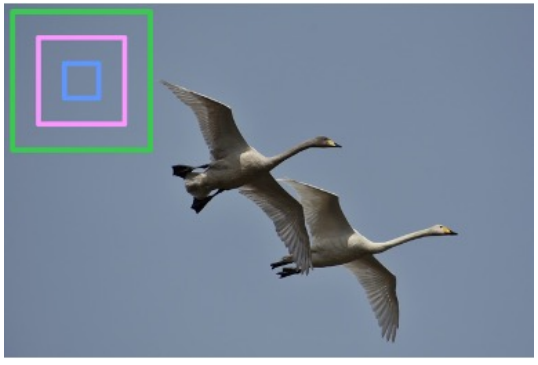




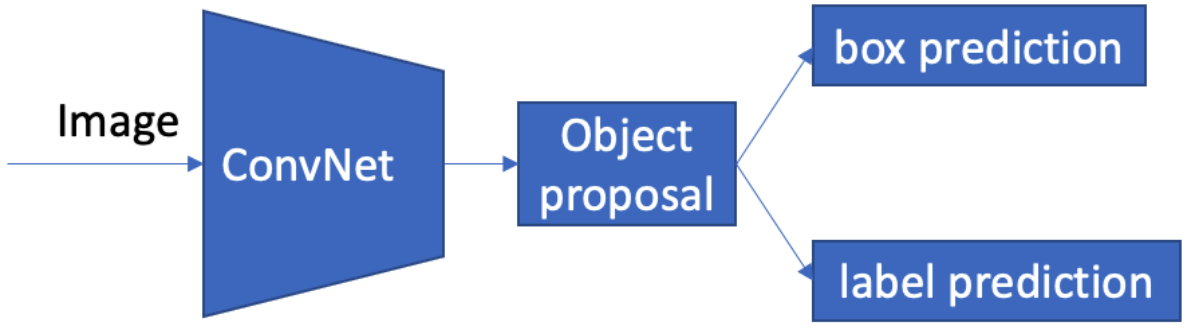
"Default boxes" also go by the name *anchor boxes* or *priors*.

Naïve (Sliding window approach) for detecting multiple objects

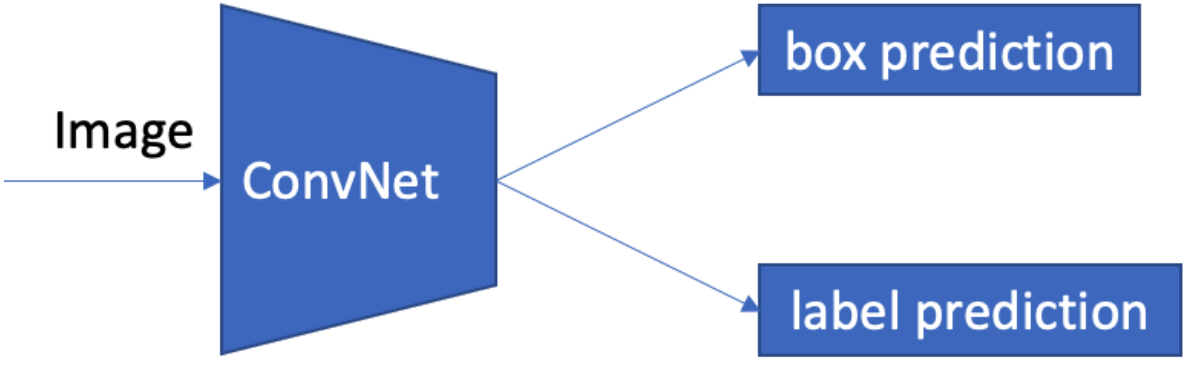
- Slide (multiple-sized) windows across the image
- Apply an image classifier on every location
- Extract local score-peaks
- OK for “cheap” classification methods
- Very slow for CNN classifiers



Object detection approaches



Two-stage detector



One-stage detector

Two stage detectors

- Stage 1
  - Generate multiple candidates (bounding boxes) all over the image. It's typical to make use of "anchors".
    - Selective search, Region Proposal Network (RPN)
  - Throw away the candidates without an object.
- Stage 2
  - Process each candidate independently
  - Assign a category (class) to each candidate and adjust its bounding box.

One stage detectors

- Pass (process) the image through a Neural Network.
- On the final feature map(s) of the network:
  - Slide a window
  - For each window location, predict an object class and a bounding box for each "anchor" (also called "default box" or "prior"), i.e., adjust the anchor.

One stage anchor-free detectors

- Same as one-stage detectors, but instead of adjusting anchor, directly predict:
  - Top-left and bottom-right corners
  - And / Or the centre of the object

R-CNN



We will start with R-CNN and see how it incorporated more neural networks to get better and faster.

R-CNN



The 1^{st} step is to run Selective Image Search on the image to get some object proposals. This is a non-deep-learning algorithm that runs on the CPU and generates some object proposals.

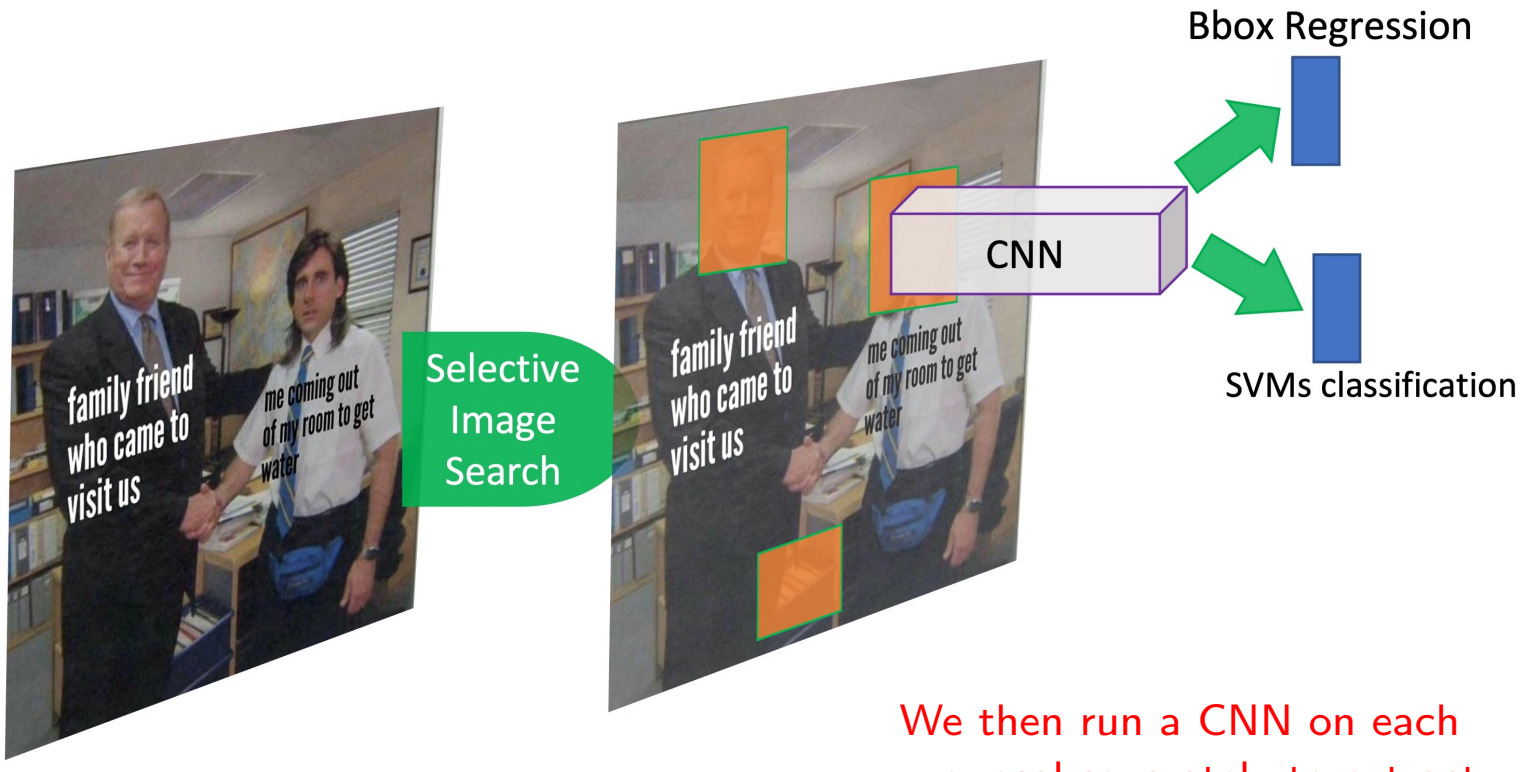
R-CNN



Let's say we get these 3 object proposals (also called ROIs - Regions of interest).

Stage 1 – Propose ROIs (Regions of Interest)

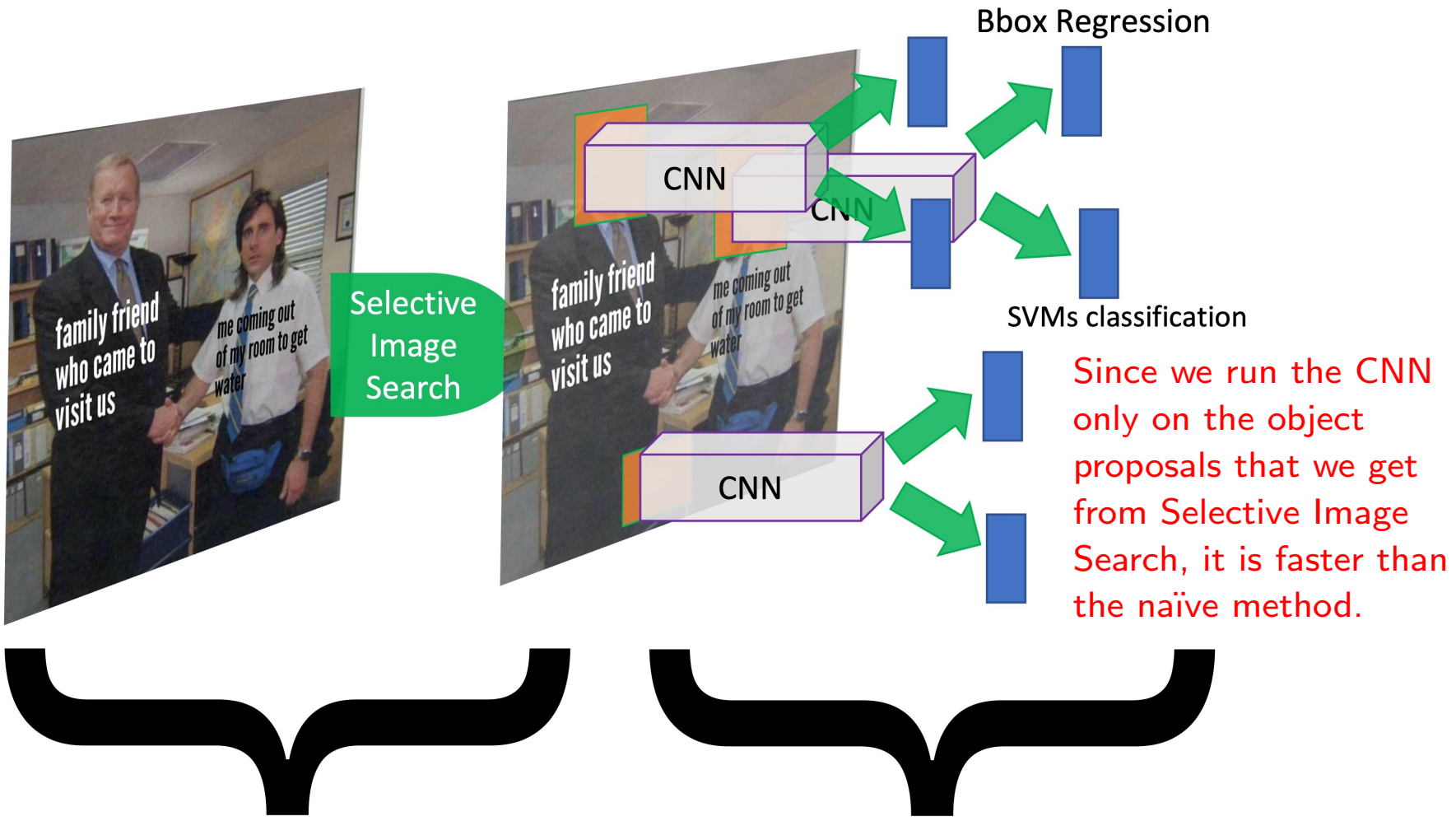
R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

We then run a CNN on each proposal separately to extract features for that proposal and finally use those features to predict the bounding box and the class label for that proposal.

R-CNN


Bbox Regression
family friend who came to visit us
me coming out of my room to get water
Selective Image Search
CNN
CNN
CNN
SVMs classification
Since we run the CNN only on the object proposals that we get from Selective Image Search, it is faster than the naïve method.
Stage 1 – Propose ROIs (Regions of Interest)
Stage 2 – Classify and adjust bbox

Fast R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

Stage 1 in Fast R-CNN remains the same. But stage 2 is where gain speed and performance by running a CNN on the whole image instead of running one independently on each proposal.

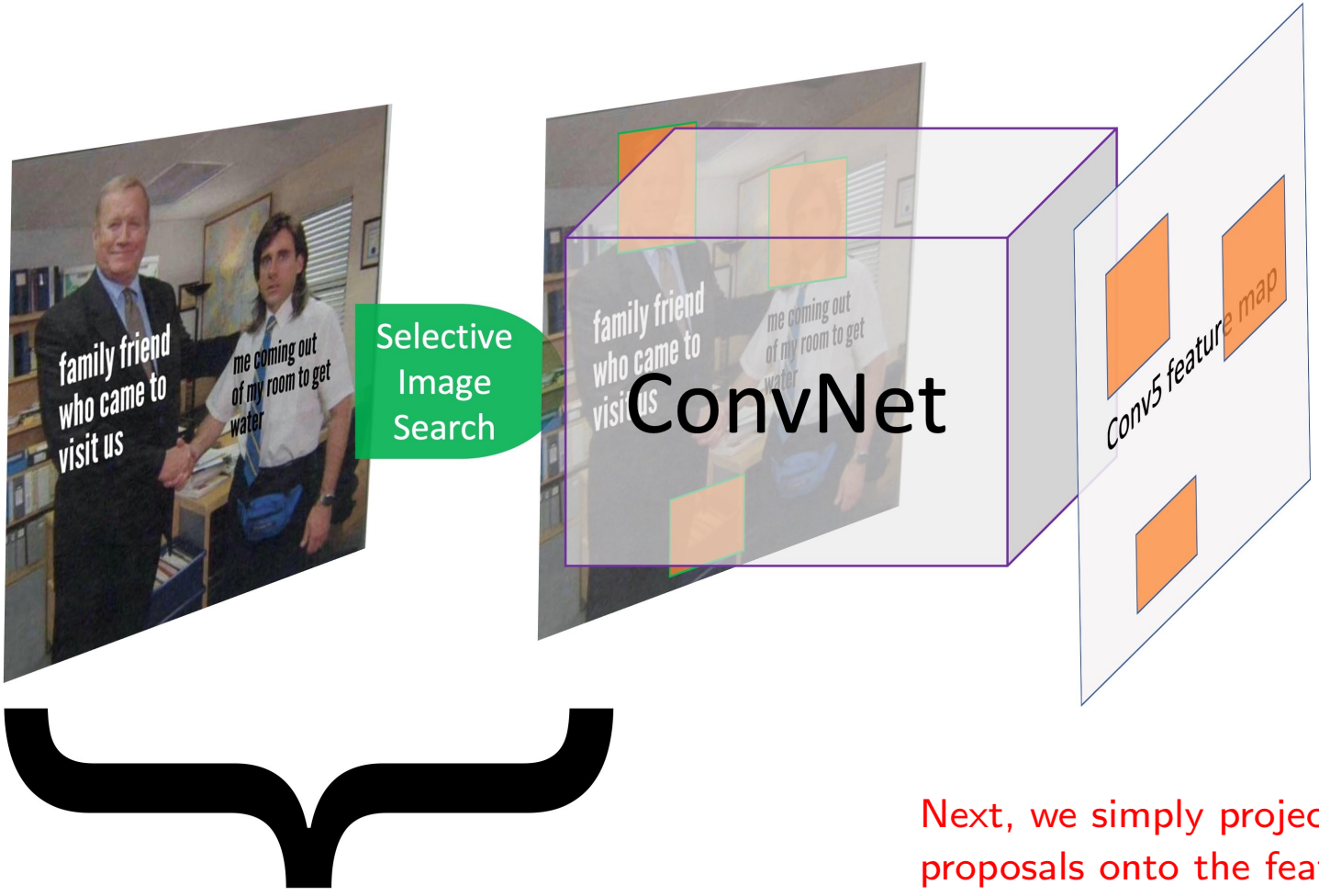
Fast R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

We pass the image through a CNN and take the feature maps from the last layer of the CNN.

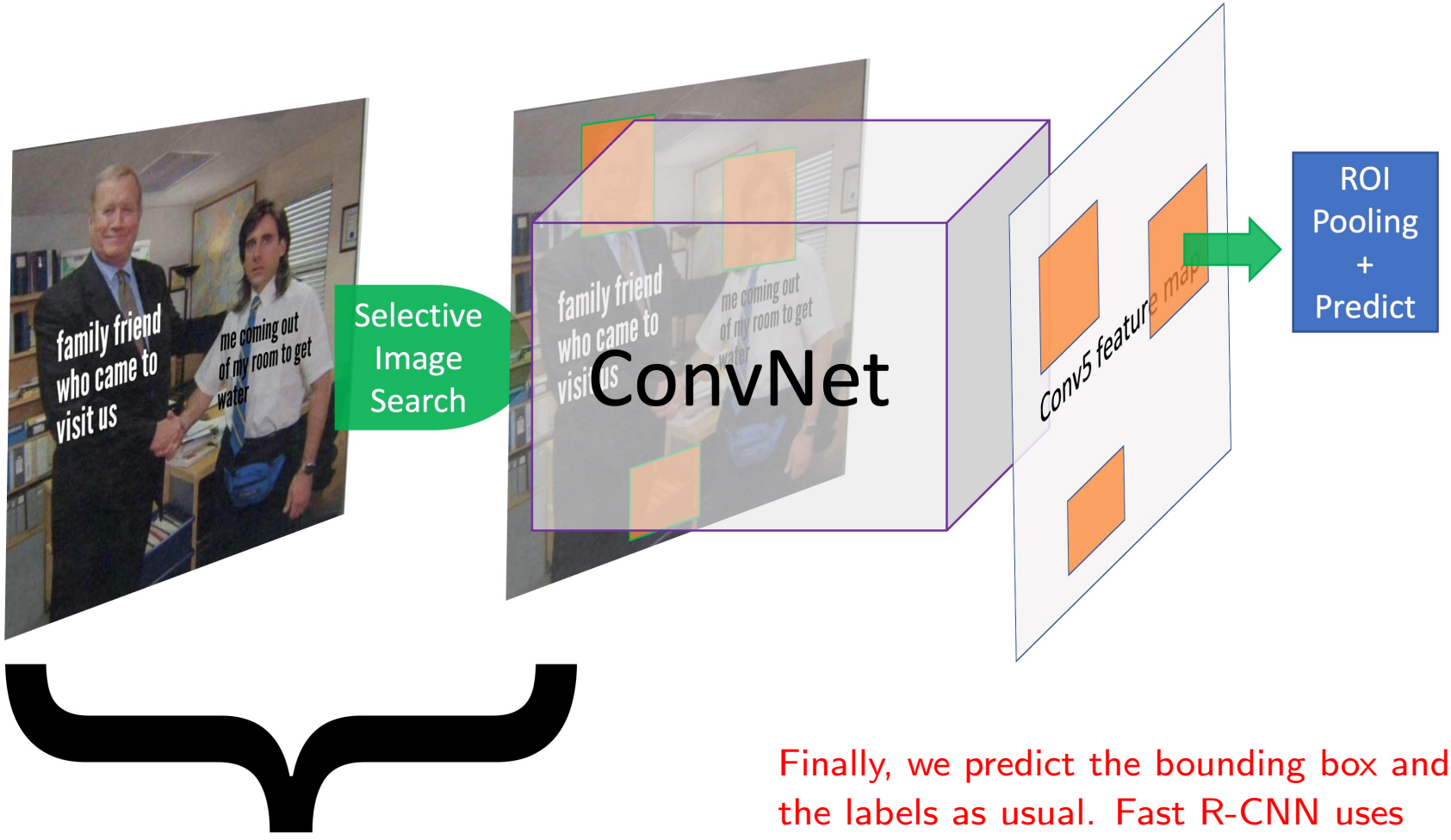
Fast R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

Next, we simply project the object proposals onto the feature map extracted from the ConvNet. Note that the extracted feature map usually has a lower resolution than the input image.

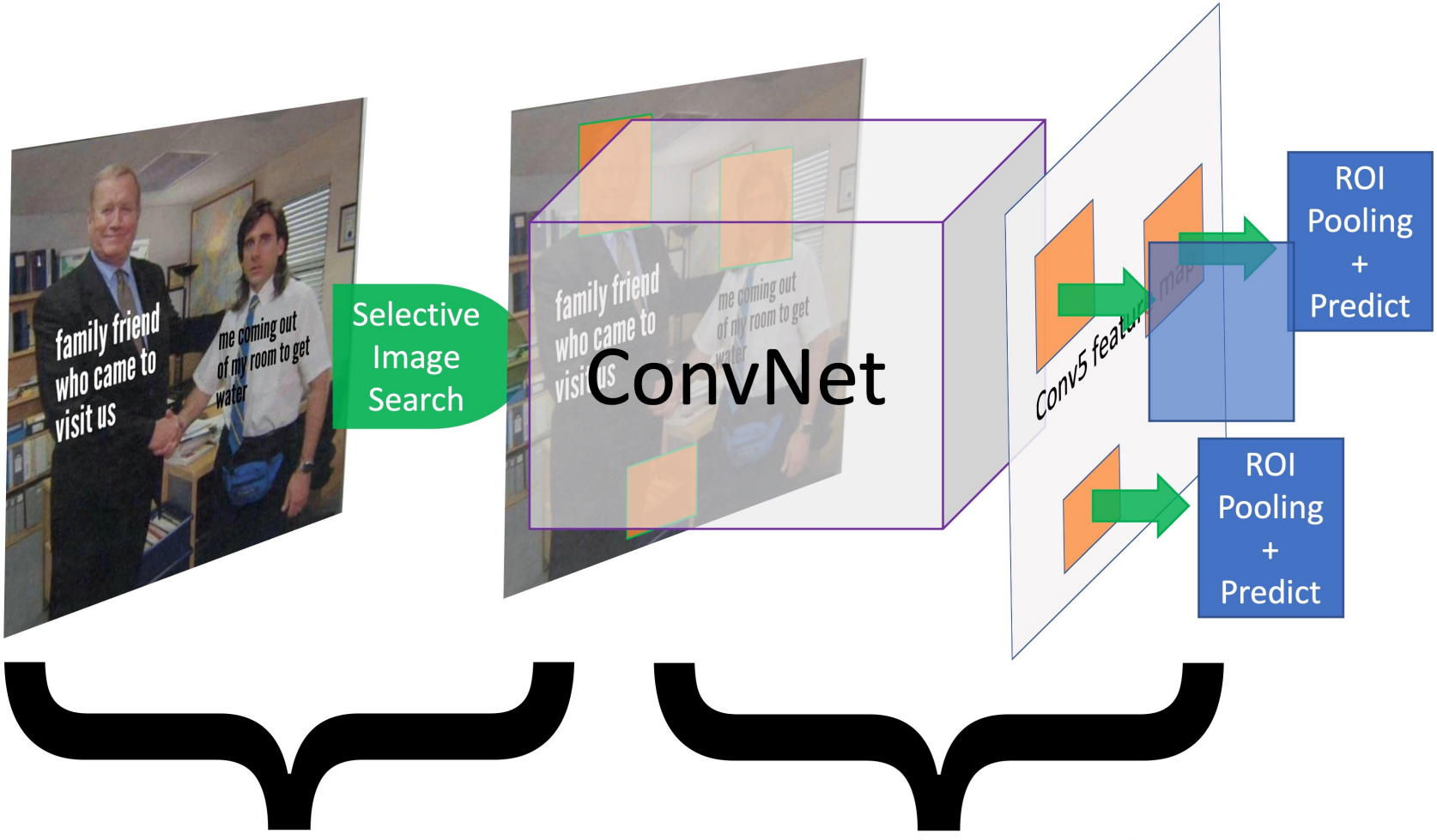
Fast R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

Finally, we predict the bounding box and the labels as usual. Fast R-CNN uses something called "ROI Pooling" before making a prediction. It's a simple operation that maps the proposals to a fixed size, but we are going to skip over those details.

Fast R-CNN



Stage 1 – Propose ROIs (Regions of Interest)

Stage 2 – Classify and adjust bbox

Faster R-CNN

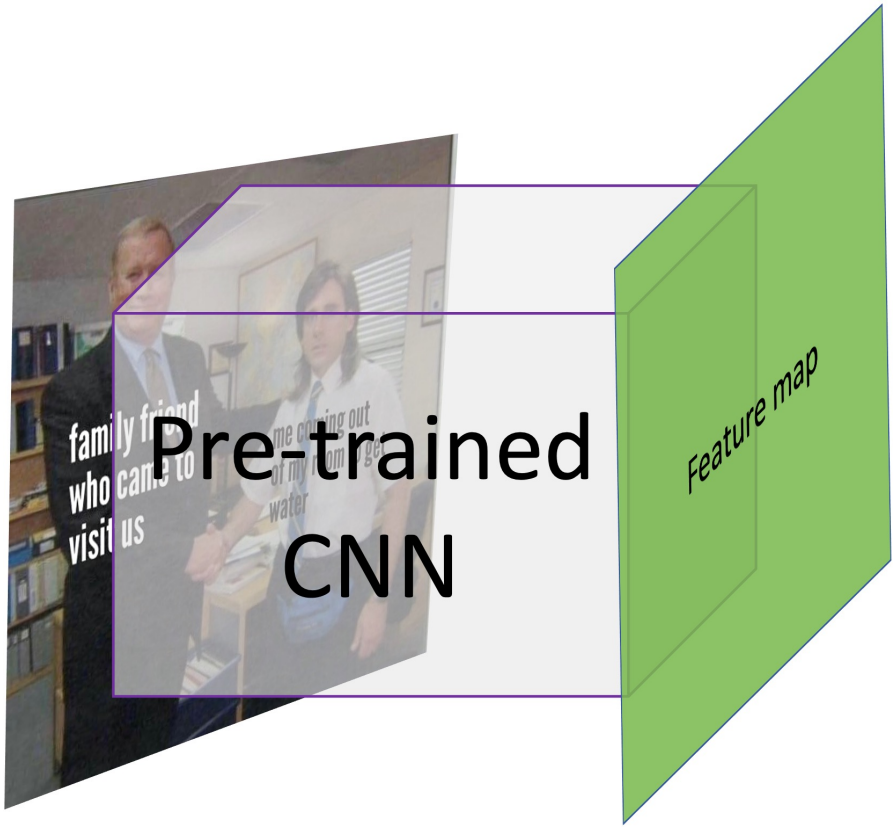


The bottleneck in Fast R-CNN is its Stage 1. It uses Selective Image Search, which runs on the CPU and is not fast enough. Faster R-CNN replaces it with two CNNs:
1) a pre-trained CNN to extract features
2) a CNN (called Region Proposal Network or RPN for short) for making the actual proposals based on the extracted features.

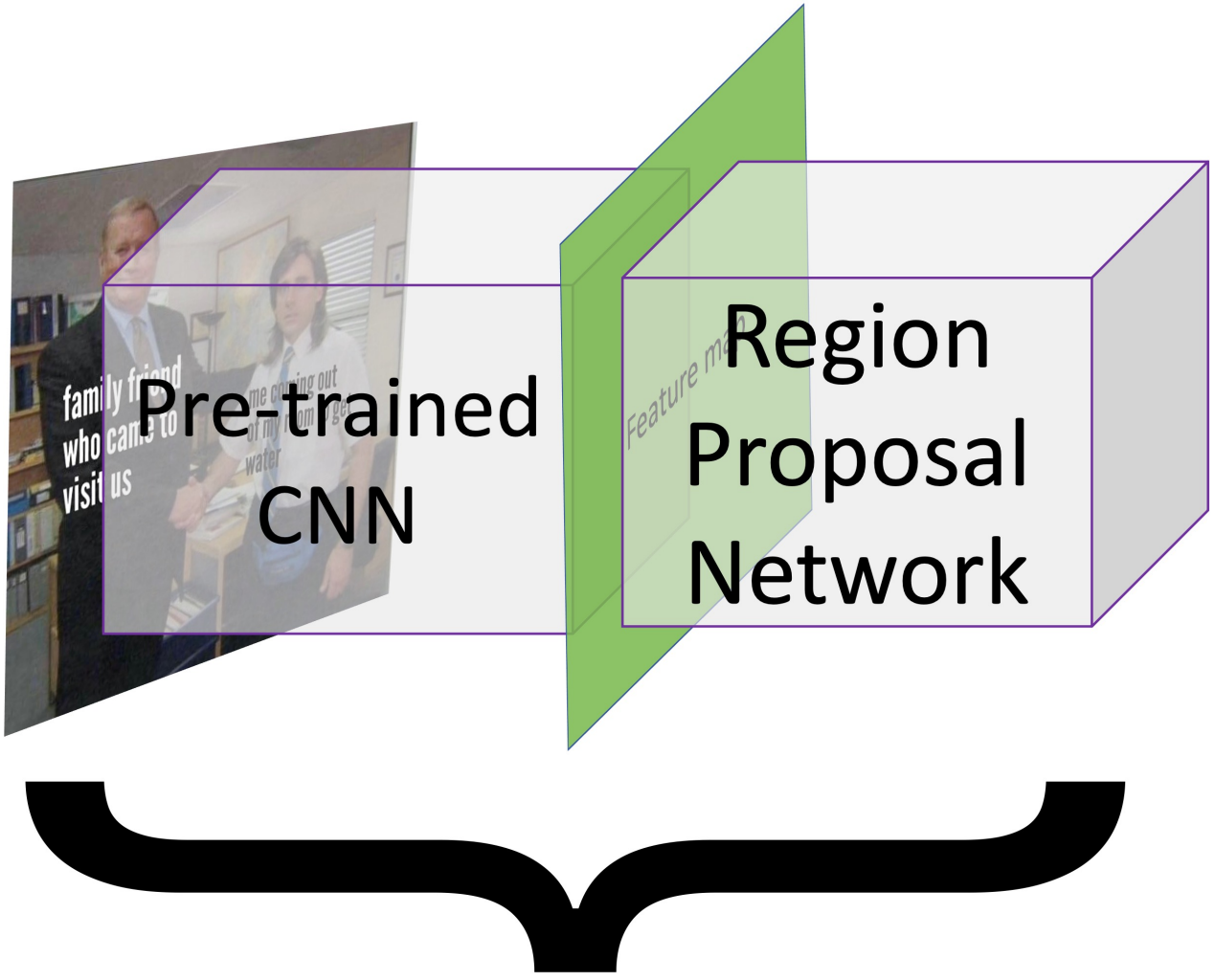
Faster R-CNN



Faster R-CNN

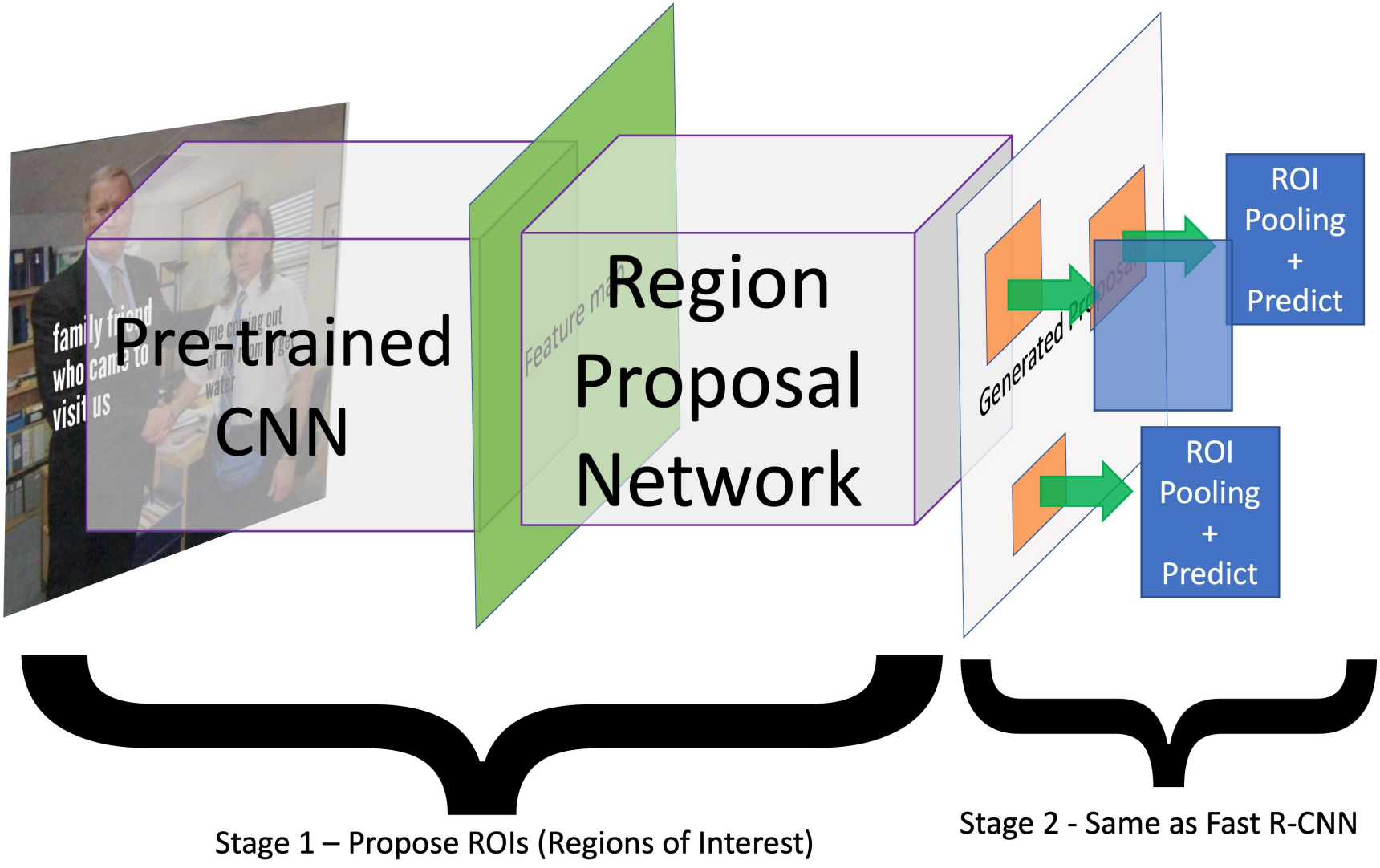


Faster R-CNN

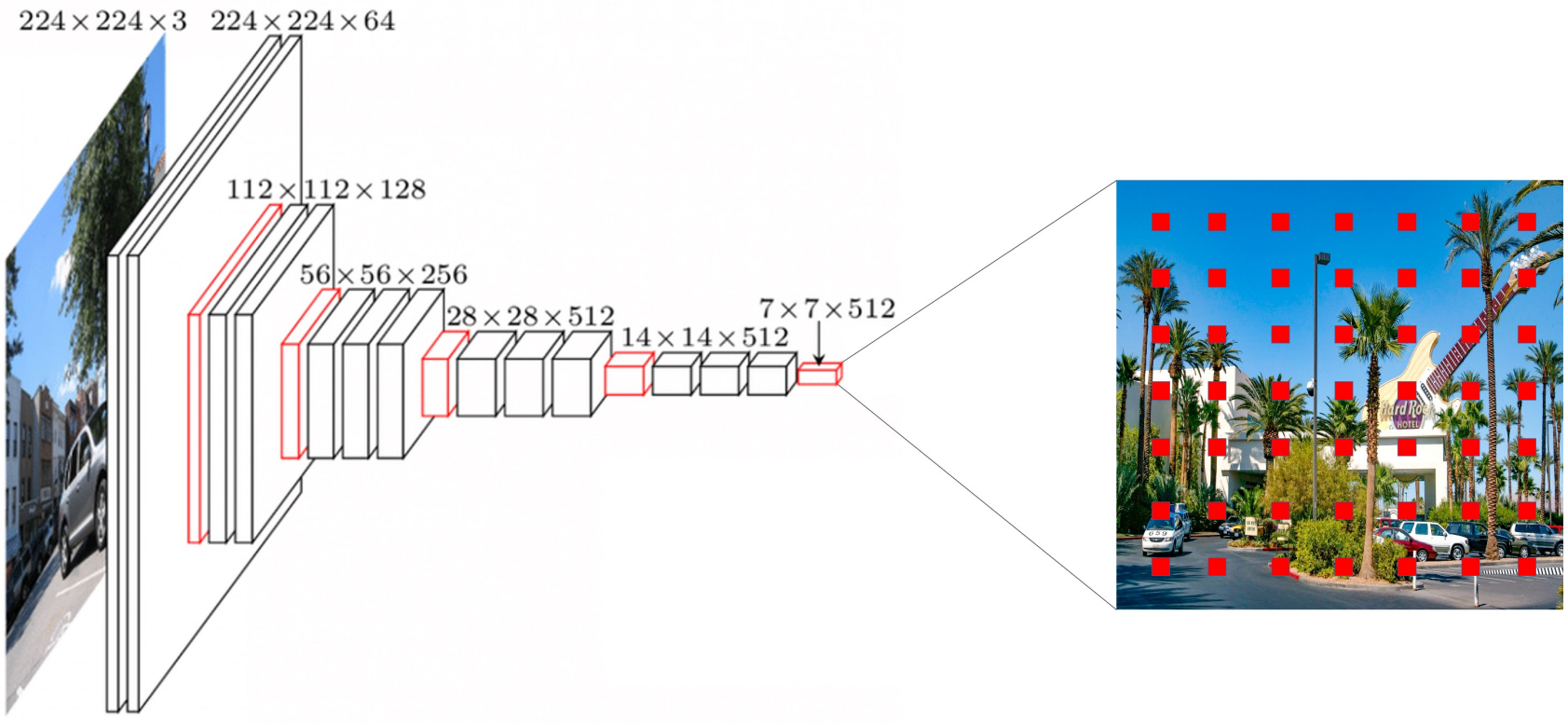


Stage 1 – Propose ROIs (Regions of Interest)

Faster R-CNN


Pre-trained CNN
Region Proposal Network
ROI Pooling + Predict
ROI Pooling + Predict
Stage 1 – Propose ROIs (Regions of Interest)
Stage 2 - Same as Fast R-CNN

Generating candidates using anchors

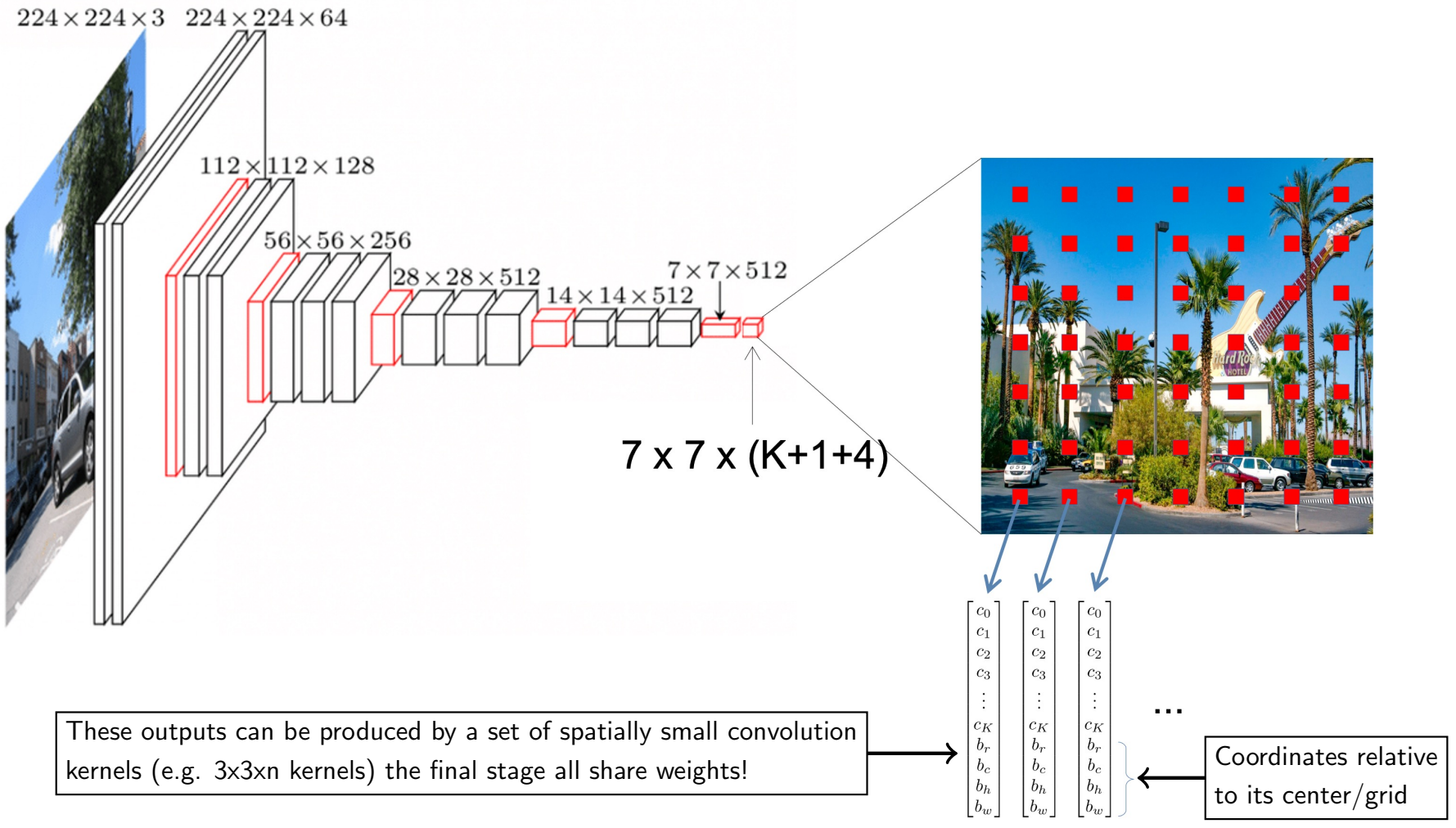


Your network already produces features that are spatially distributed

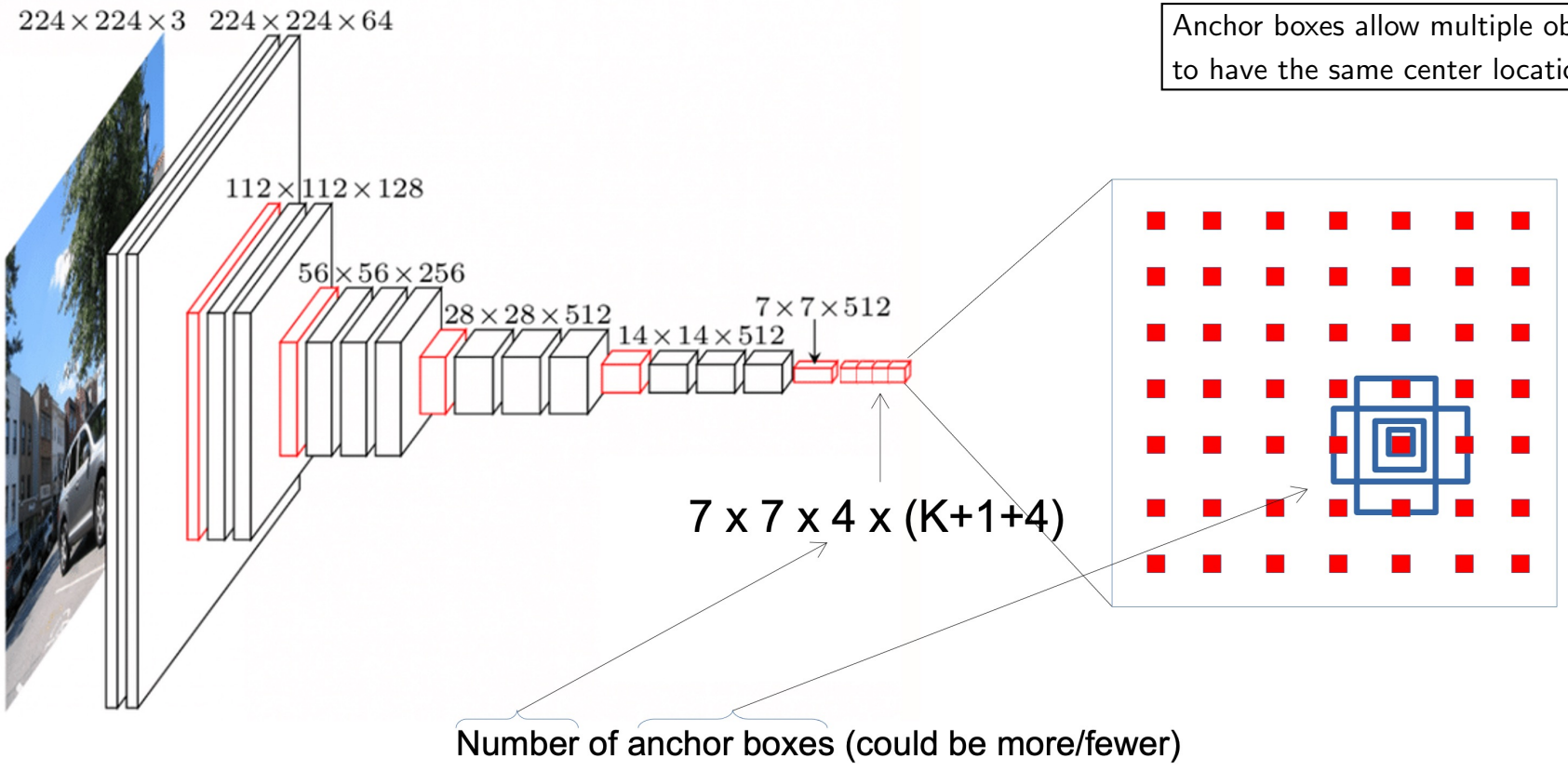
Generating candidates using anchors

- This lecture comes with a Jupyter notebook to help you understand the concept of anchors better:
- https://www.uio.no/studier/emner/matnat/ifi/IN3310/v25/forelesning/anchors.ipynb

Generating candidates using anchors



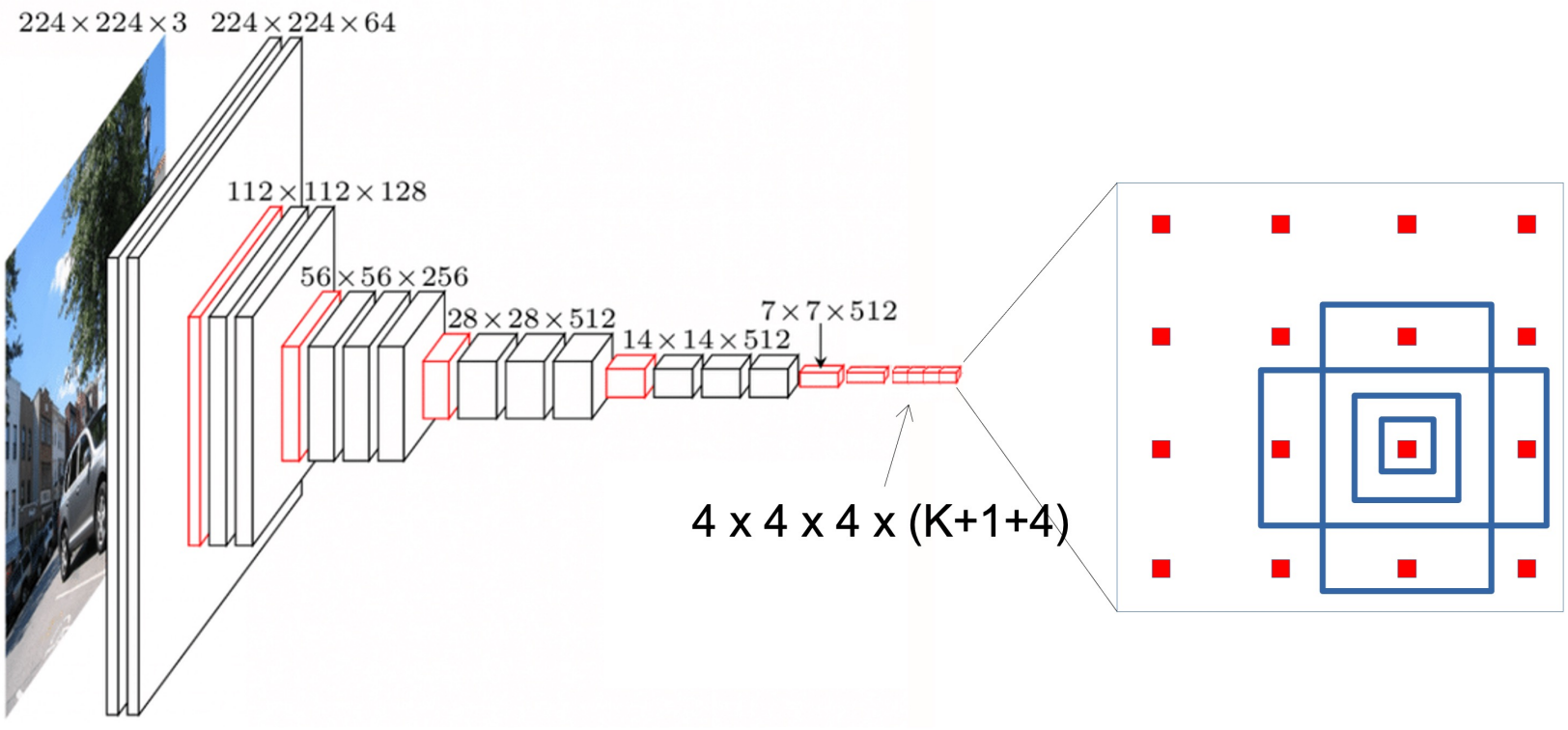
Generating candidates using anchors



Anchor boxes allow multiple objects to have the same center location

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

Generating candidates using anchors



Anchors on lower resolution feature maps can find bigger objects

Likely *fewer* bigger boxes needed

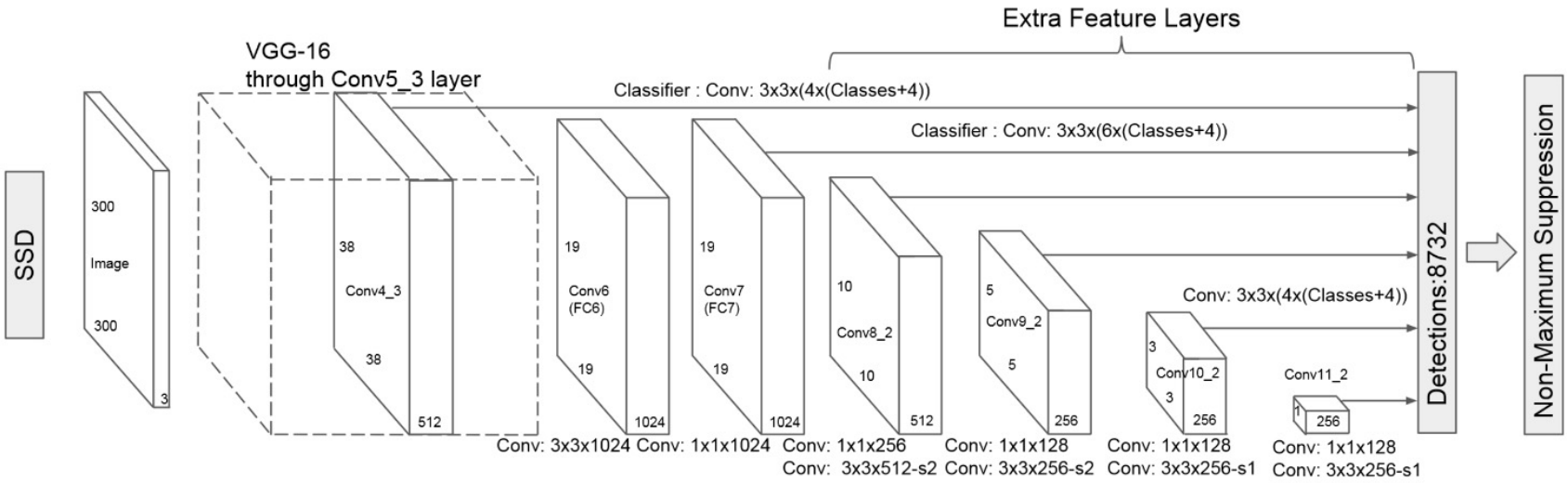
.. and, if still using 3x3 convolutions, the bigger boxes is predicted using a larger image context

Problem 1: Different object sizes

- Objects can be big or small across images or even within an image.
- Anchors of varying sizes help but anchors at a location are limited by that location's receptive field size.
- We'll look at two solutions (there are many):
  - Single Shot Multibox Detector (SSD)
  - Feature Pyramid Network (FPN)

Single-shot multibox detector (SSD)

- One stage detector



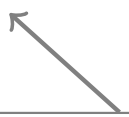
Liu, Wei, et al. "SSD: Single shot multibox detector", ECCV, 2016.

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").

Feature-pyramid networks (FPN) I/II

- Originally proposed as a two-stage detector.
  - Used as a backbone in one-stage detectors like EfficientDet.
- Deeper layers $\rightarrow$ 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!



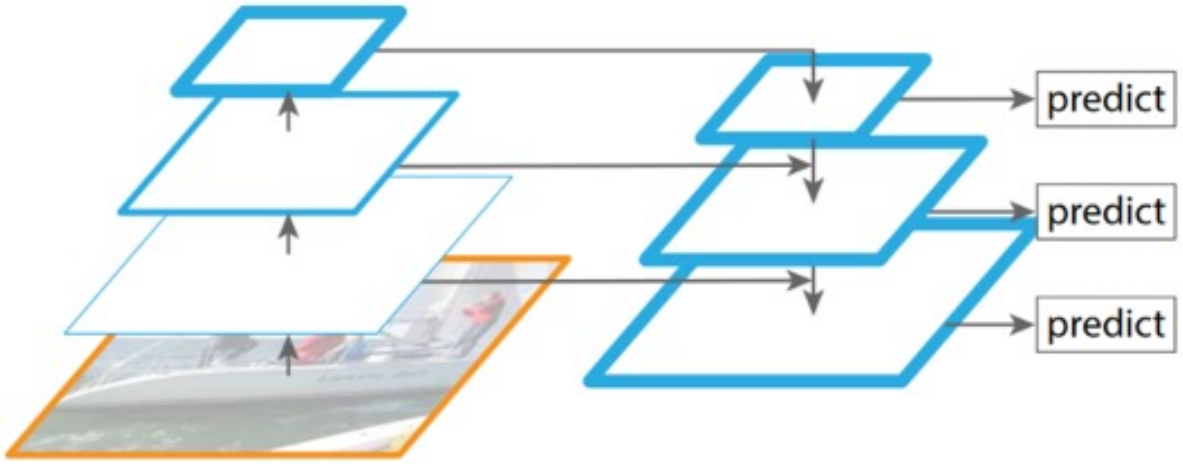
Note that there are many different ways of achieving this, however the "FPN" described on the next slides is easily implemented, fast and does wonders in practice..

Context matters!

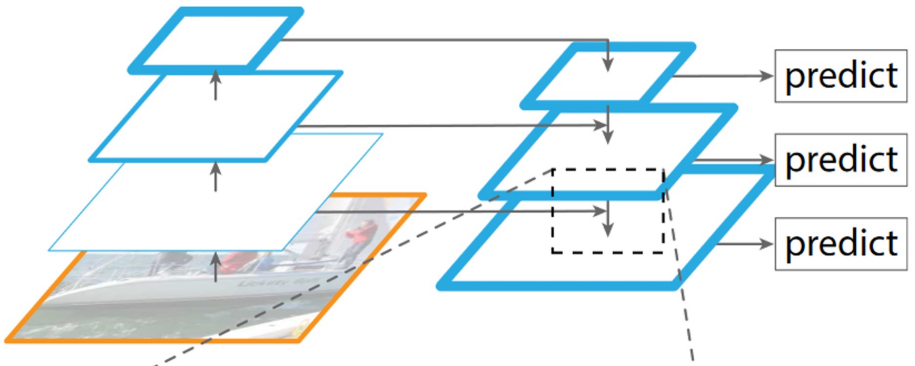




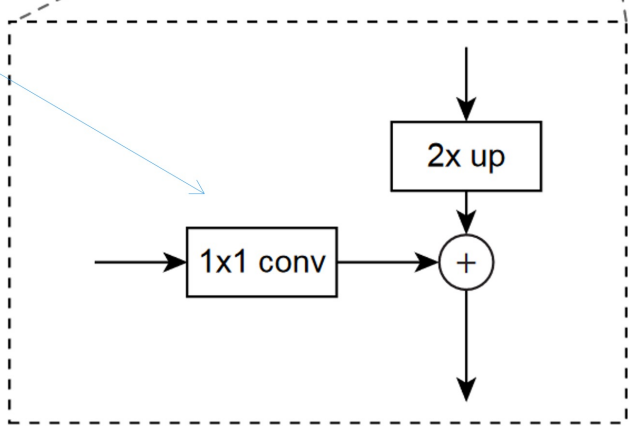
Feature-pyramid networks (FPN) II/II



Feature-pyramid networks (FPN) II/II



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

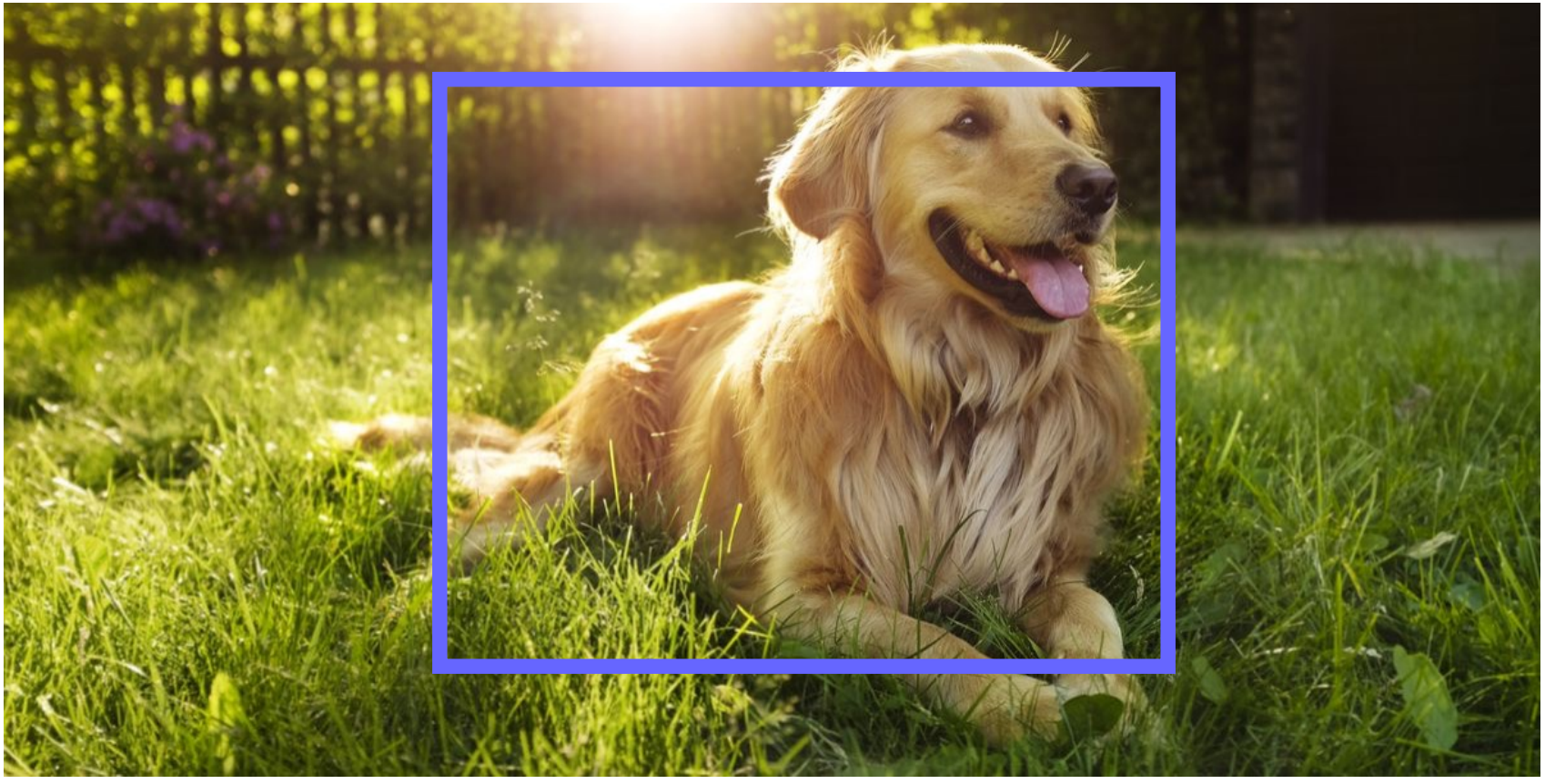


Lin et al. "Feature Pyramid Networks for Object Detection", CVPR, 2017.

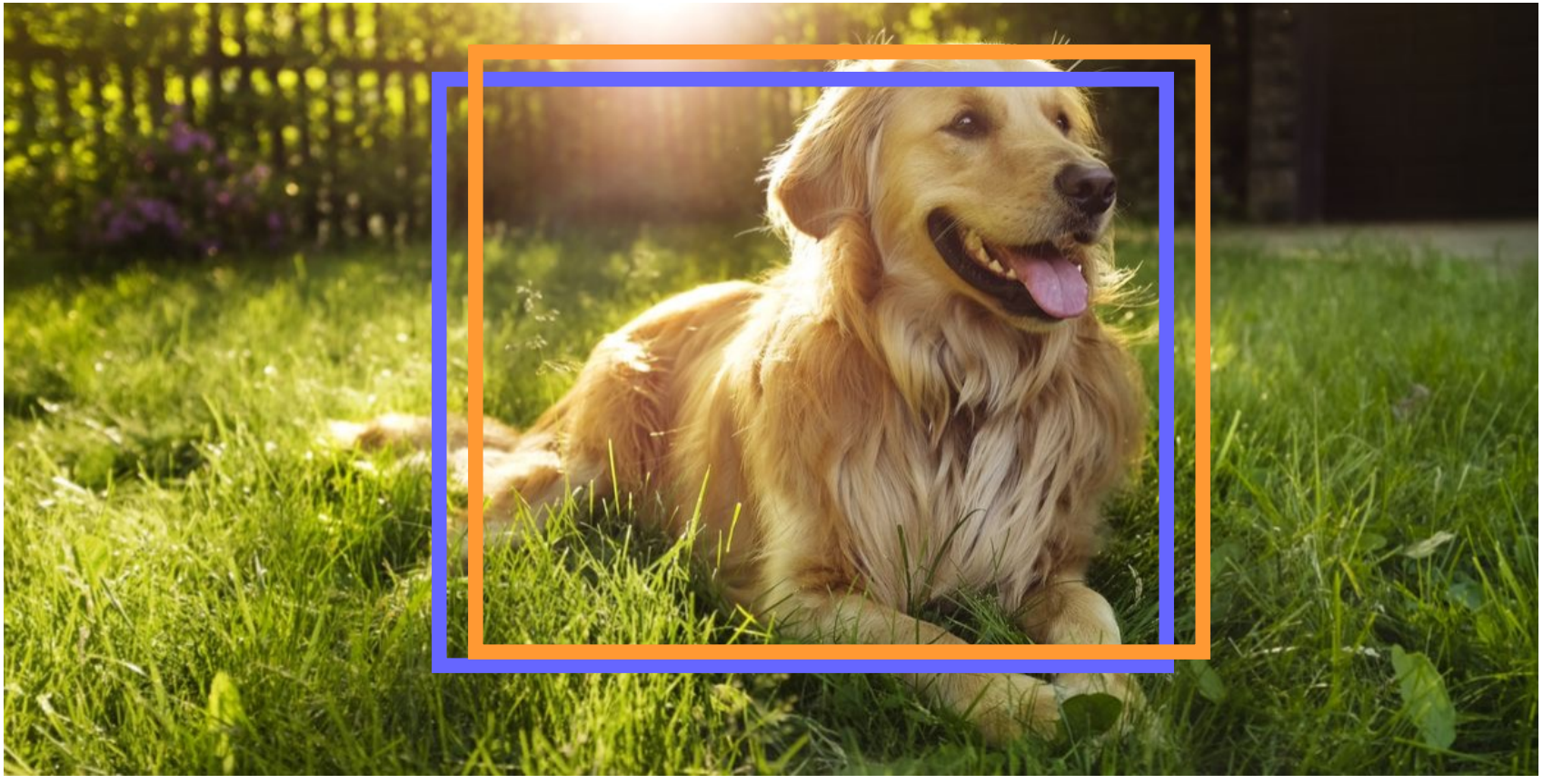
Problem 2: Multiple overlapping predictions



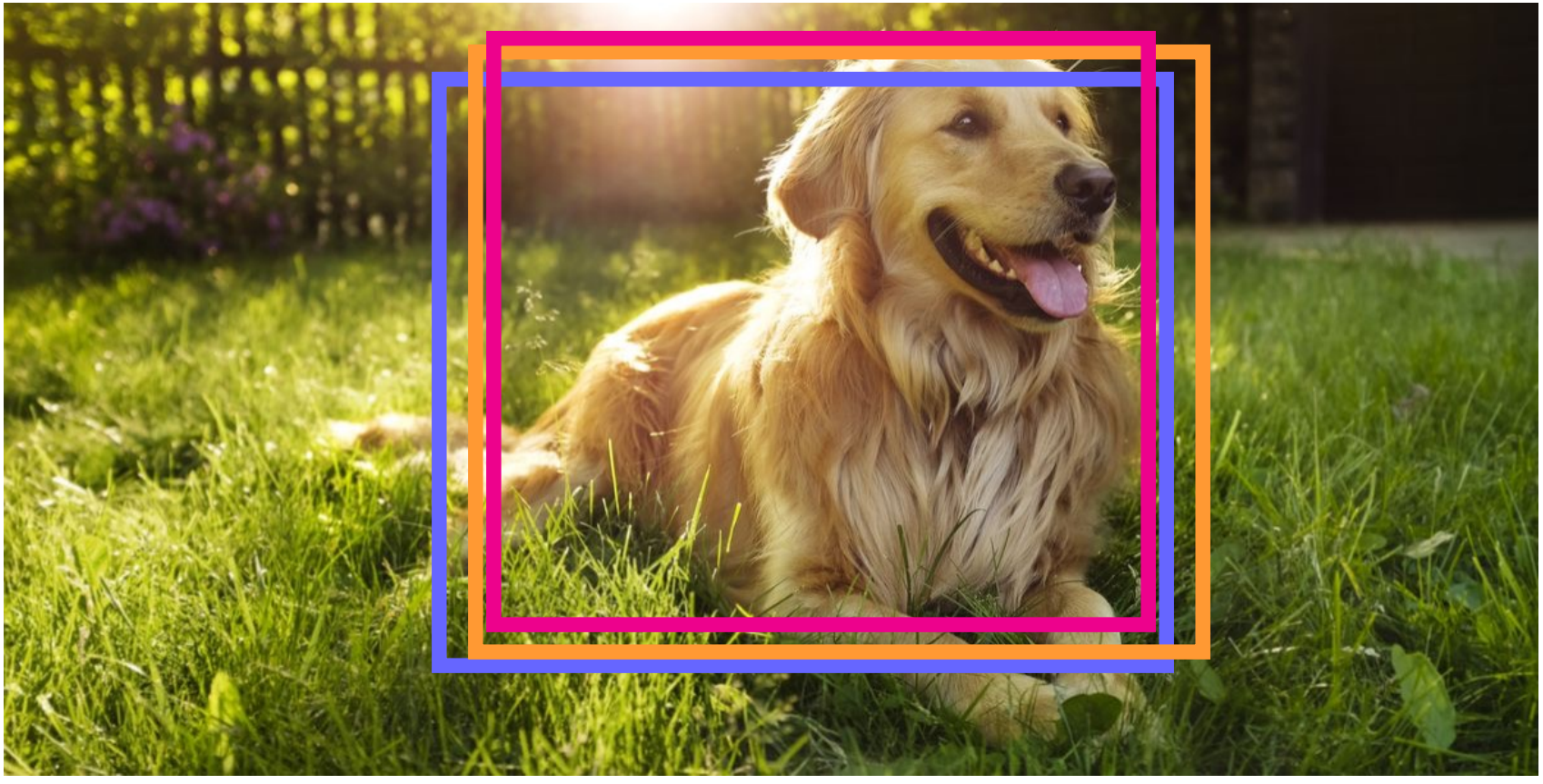
Problem 2: Multiple overlapping predictions



Problem 2: Multiple overlapping predictions



Problem 2: Multiple overlapping predictions



Problem 2: Multiple overlapping predictions

- An object in an image can have many overlapping boxes that cover it.
- How do we know which ones to throw away?
  - Use Non-Maximum Suppression (NMS).

Non-max suppression I/III

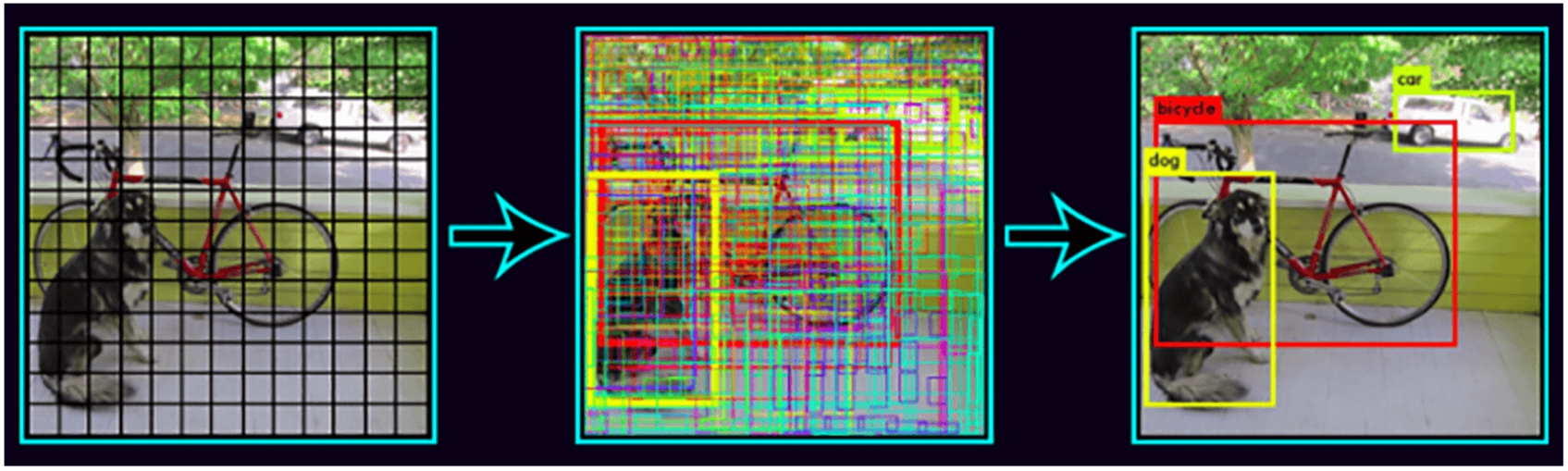
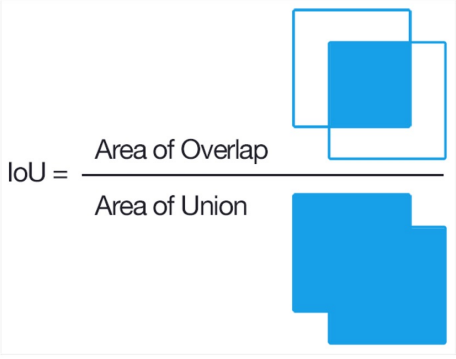


Illustration from the YOLO 2015 paper

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,...,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 the highest value of c_i (or some similar criterion)
    - Remove all regions that overlap with R_{max} (e.g. with iou > 0.5) from U_i
    - Move R_{max} from U_i to K_i

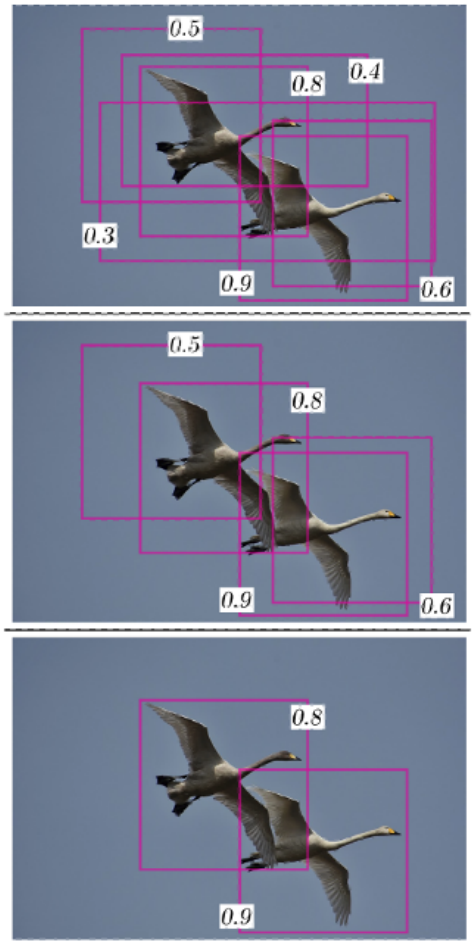


Figure 29: Top: Original. Middle: Too low c_0 removed. Bottom: $iou > 0.5$ removed. Image source: https://www.pixabay.com

Notes on how to train I/II

How to match anchors with ground truth boxes?

Matching strategy used by SSD:

```
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/II

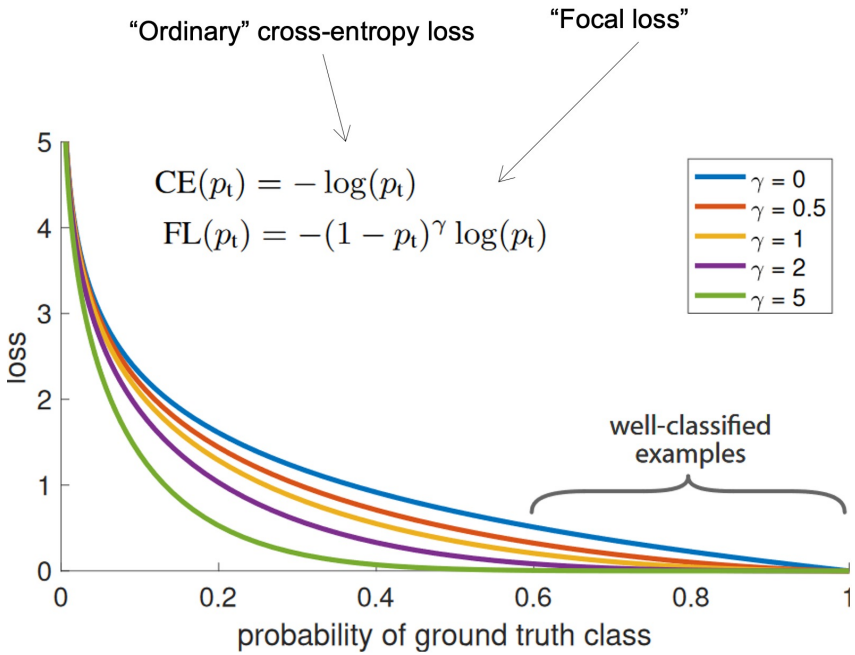
- Data augmentation (as always..)
  - Create more (and plausible so) data; cropping, resizing, mirroring, photometric distortions, ...

Problem 3: Too many background predictions

- Often many more anchor-boxes without a match $\rightarrow$ imbalanced dataset
  - Hard Negative Mining
    - Select only *the most difficult* background patches (lowest background score) when computing loss.
    - See OHEM (Online Hard Example Mining) for instance.
  - Change loss function to downscale importance of highly certain background patches $\rightarrow$ "focal loss" (see next slide)
    - Mostly used by one-stage detectors

Focal Loss

- "Focal loss. . . focuses training on a sparse set of hard examples"
- Vigour 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



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

Let's say we have many background patches which we are quite certain about being correctly labelled (p $\approx$ 0.7).

Using the CE-loss, we can substantially reduce this loss by going from "quite certain" to "very certain" (p $\approx$ 0.9), not so for the FL-loss.

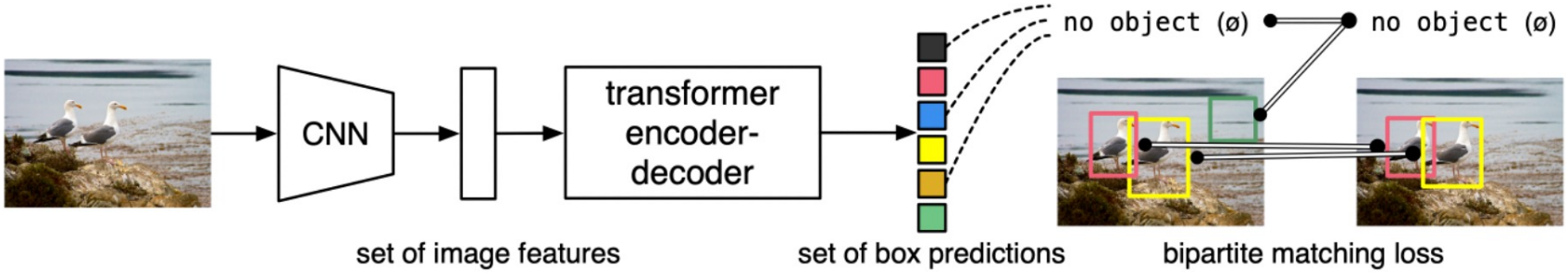
Lin et al.. "Focal Loss for Dense Object Detection", ICCV, 2017.

Outline

1. Single-instance detection and localization
2. Object detection (multiple objects)
   - 2.1 Two stage detectors and one stage detectors
   - 2.2 A popular two-stage detector: Faster R-CNN
   - 2.3 Generating candidates using anchors
   - 2.4 Problem 1: Different object sizes
   - 2.5 Problem 2: Multiple overlapping predictions
   - 2.6 Notes on how to train
   - 2.7 Problem 3: Too many background predictions
3. DEtection TRansformer (DETR) for object detection
4. Performance and evaluation metrics

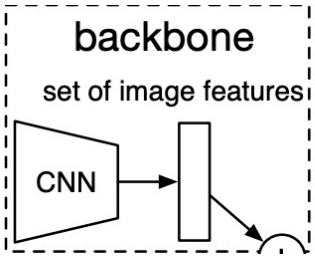
Object detection with transformers: DETR

- Simple object detection pipeline: directly output a set of boxes from a Transformer.
- Anchor-free approach, i.e., no pre-defined anchors are used.
- No Non-Maximum Suppression (NMS) used. It is learned by the transformer automatically.
- Match predicted boxes to ground truth boxes with bipartite matching.



Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

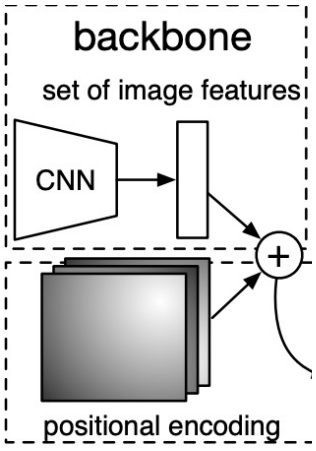
Object detection with transformers: DETR



A backbone ConvNet (like ResNet50) is used to extract features from the input.

1x1 conv layer is used to reduce the number of channels.

Object detection with transformers: DETR

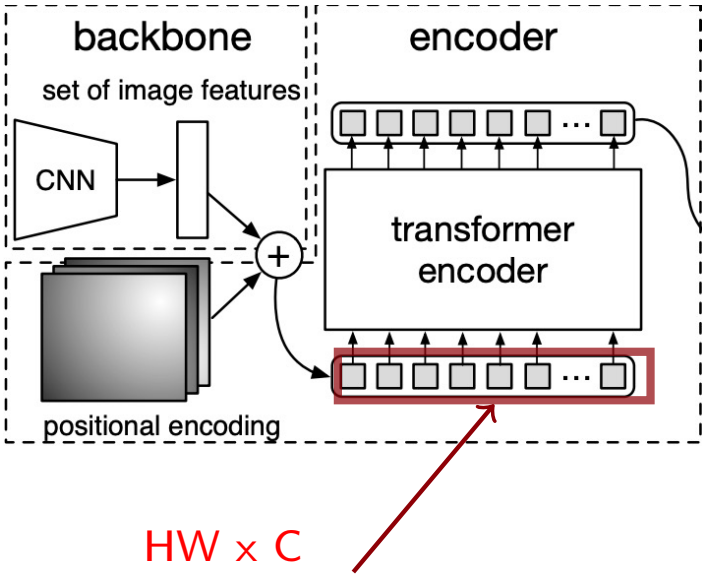


A backbone ConvNet (like ResNet50) is used to extract features from the input.

1x1 conv layer is used to reduce the number of channels.

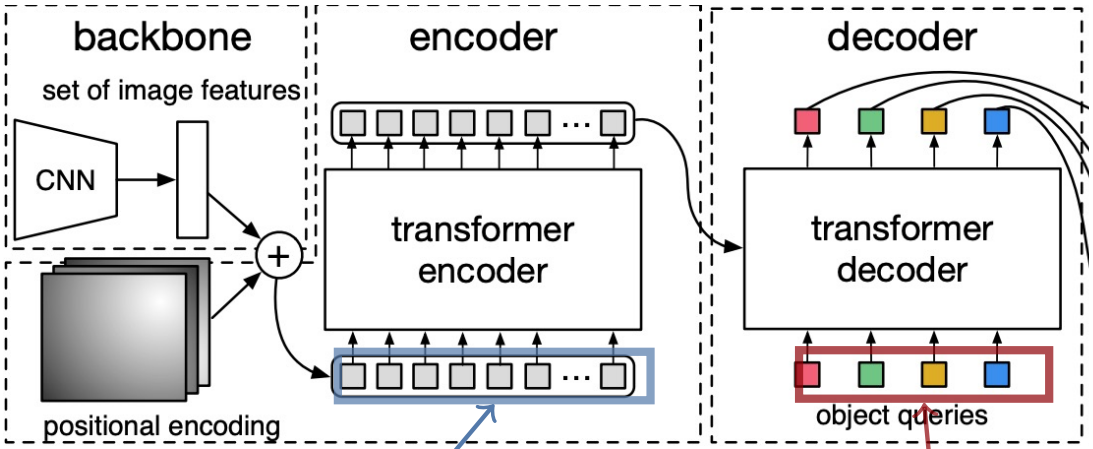
Fixed positional encodings are added to every encoder block's input

Object detection with transformers: DETR



Standard transformer encoder is used to process the flattened image features.

Object detection with transformers: DETR



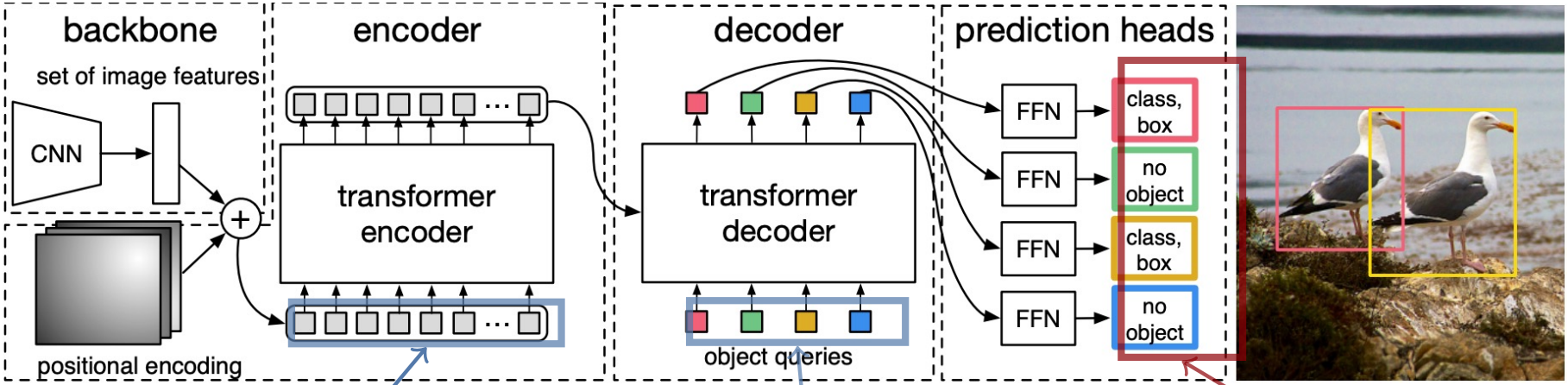
A standard transformer decoder is used to generate outputs.

Input: N object queries, encoder's output

HW x C
Flattened image features

N x C
learnable object queries
(randomly initialised)

Object detection with transformers: DETR



HW x C
Flattened image features

N x C
learnable object queries
(randomly initialised)

Each object query predicts one bounding box

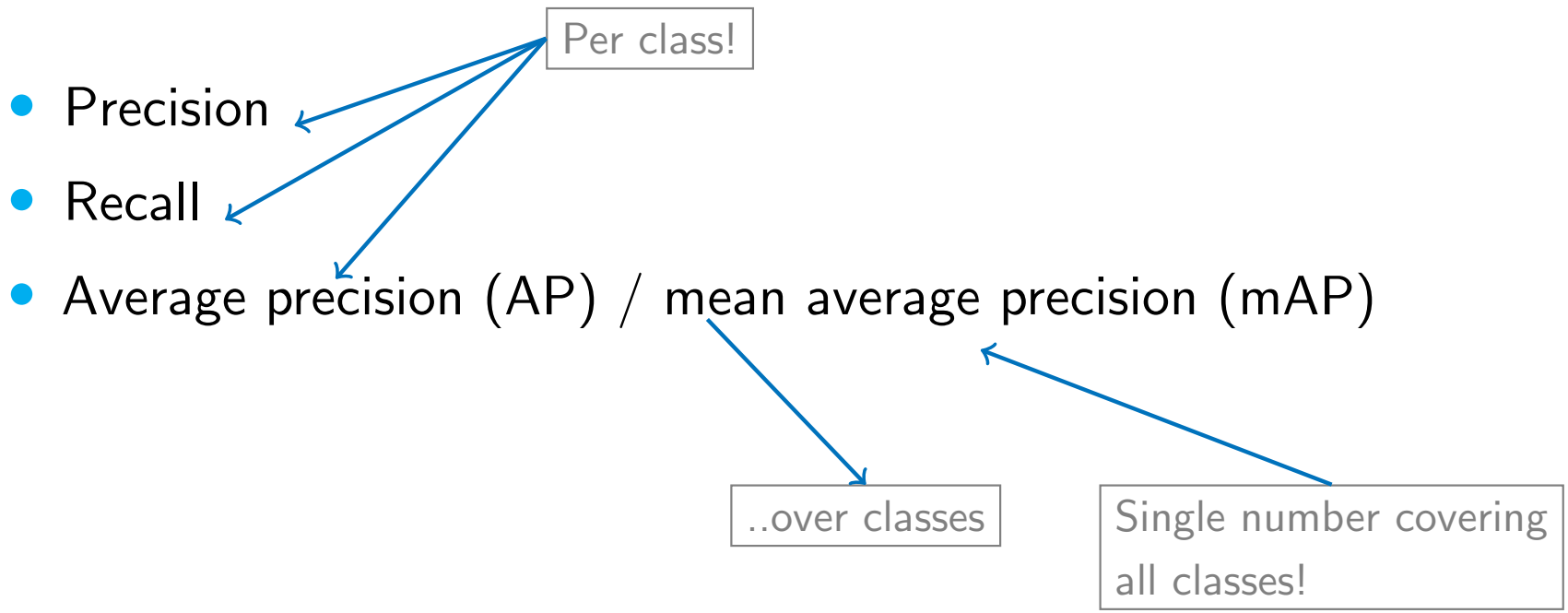
Near-infinite set of refinements..

- "Backbones"
  - VGGs / ResNets / ConvNeXts / Transformers / ..
- Pretraining
  - What data and tasks are they trained on?
- Methods/implementations
  - SSD / RetinaNet / YOLO / YOLOv2 / YOLOv8 / ..
- Many versions and combinations of concepts (adaptive anchor-boxes, layer-merges, losses, learned NMS, ...)
- Bag of Freebies – a bunch of things you can do that increase the training time but not the inference time.

Outline

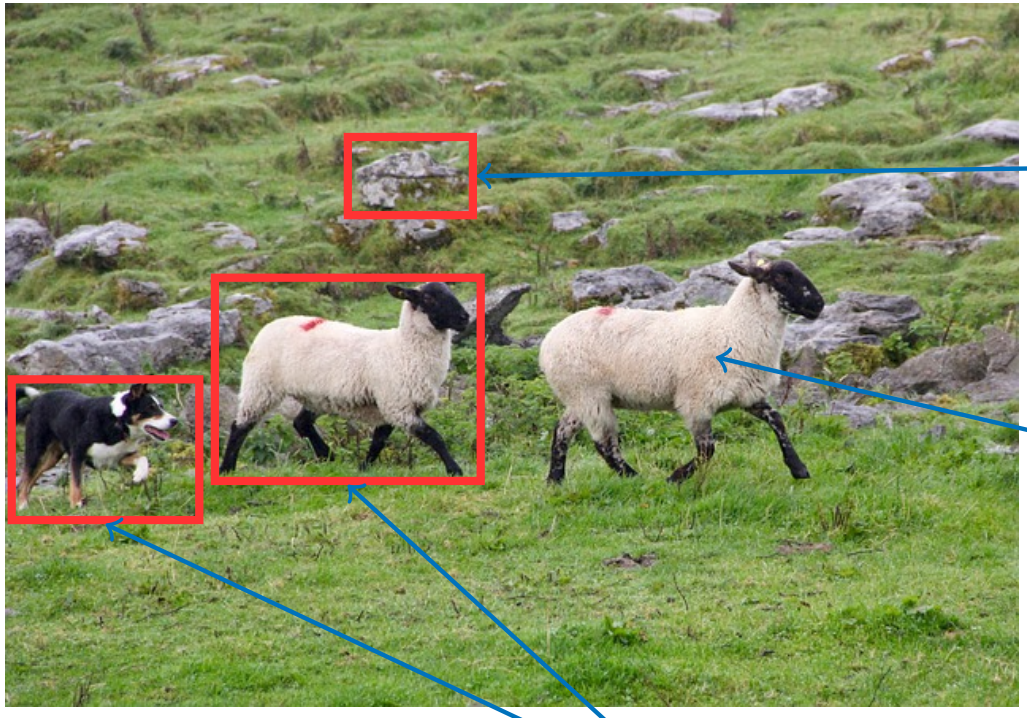
1. Single-instance detection and localization
2. Object detection (multiple objects)
   - 2.1 Two stage detectors and one stage detectors
   - 2.2 A popular two-stage detector: Faster R-CNN
   - 2.3 Generating candidates using anchors
   - 2.4 Problem 1: Different object sizes
   - 2.5 Problem 2: Multiple overlapping predictions
   - 2.6 Notes on how to train
   - 2.7 Problem 3: Too many background predictions
3. DEtection TRansformer (DETR) for object detection
4. Performance and evaluation metrics

Performance and evaluation metrics



TP, FP, FN

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.

Precision, Recall

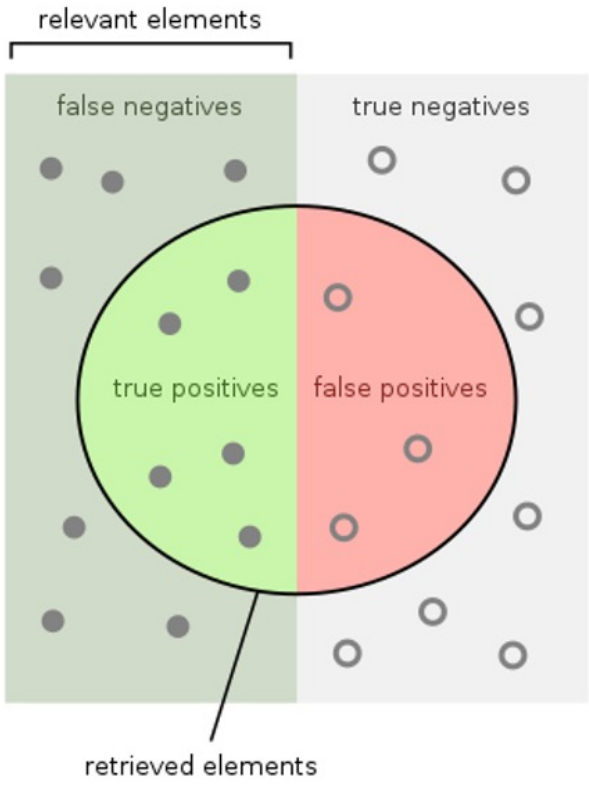
$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

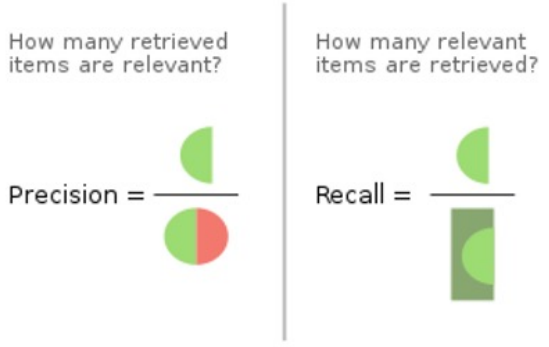
(How many of our predictions are actually true/objects)

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

(How many of the objects of interest are found)

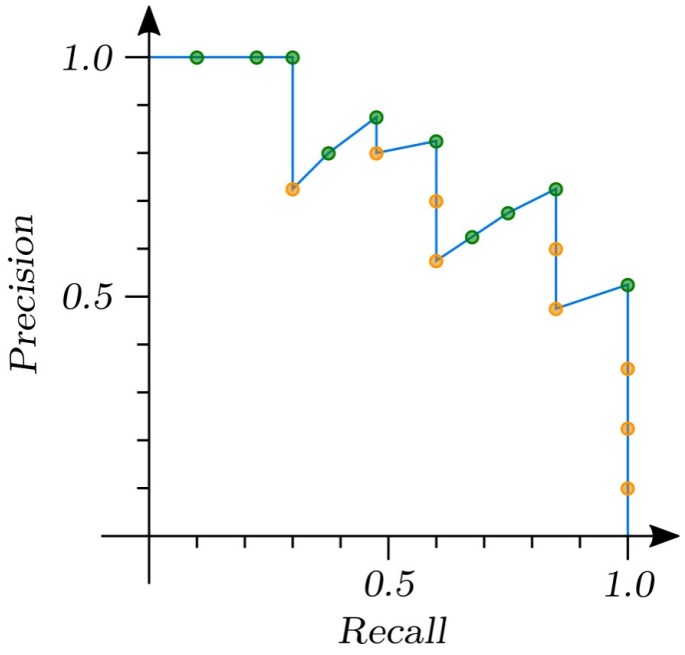
(Remember: Precision/recall is calculated **per class**!)





Precision-recall curves

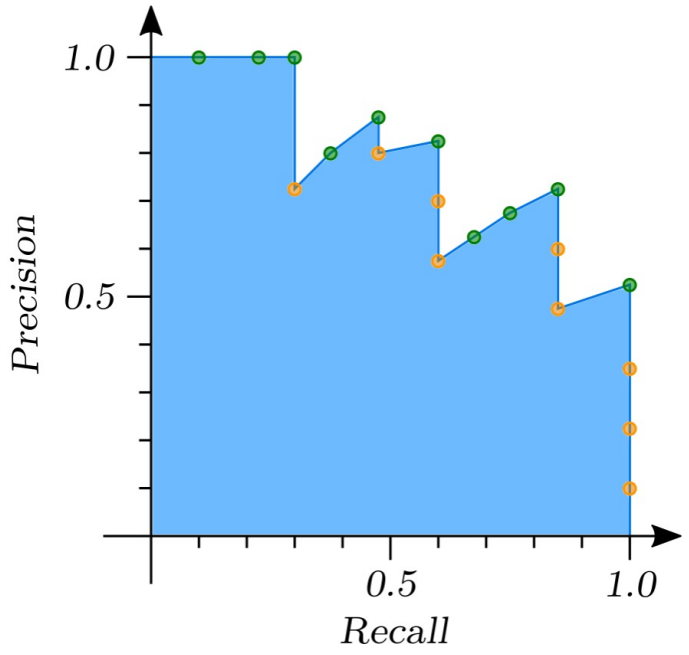
- All detected objects have a class score, c_k, for each class k
- If we alter the score-threshold for what constitutes a predicted object of a given class, the precision and recall for this class will change
- Typically, lowering the threshold gives more detections, which increases the recall but lowers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

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

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

In practice

- PyTorch/torchvision
- pytorch.org/hub/
- MMDetection, Detectron2
  - Actively developed toolboxes
  - A wealth of “backbones”, architectures and techniques
- Tensorflow object detection API

Other suggested resources

- University of Michigan's lectures on object detection:
  - https://www.youtube.com/watch?v=TB-fdISzpHQ
  - 1^{st} half of
    https://www.youtube.com/watch?v=9AyMR4IhSWQ