



MULTIPLE LABELS

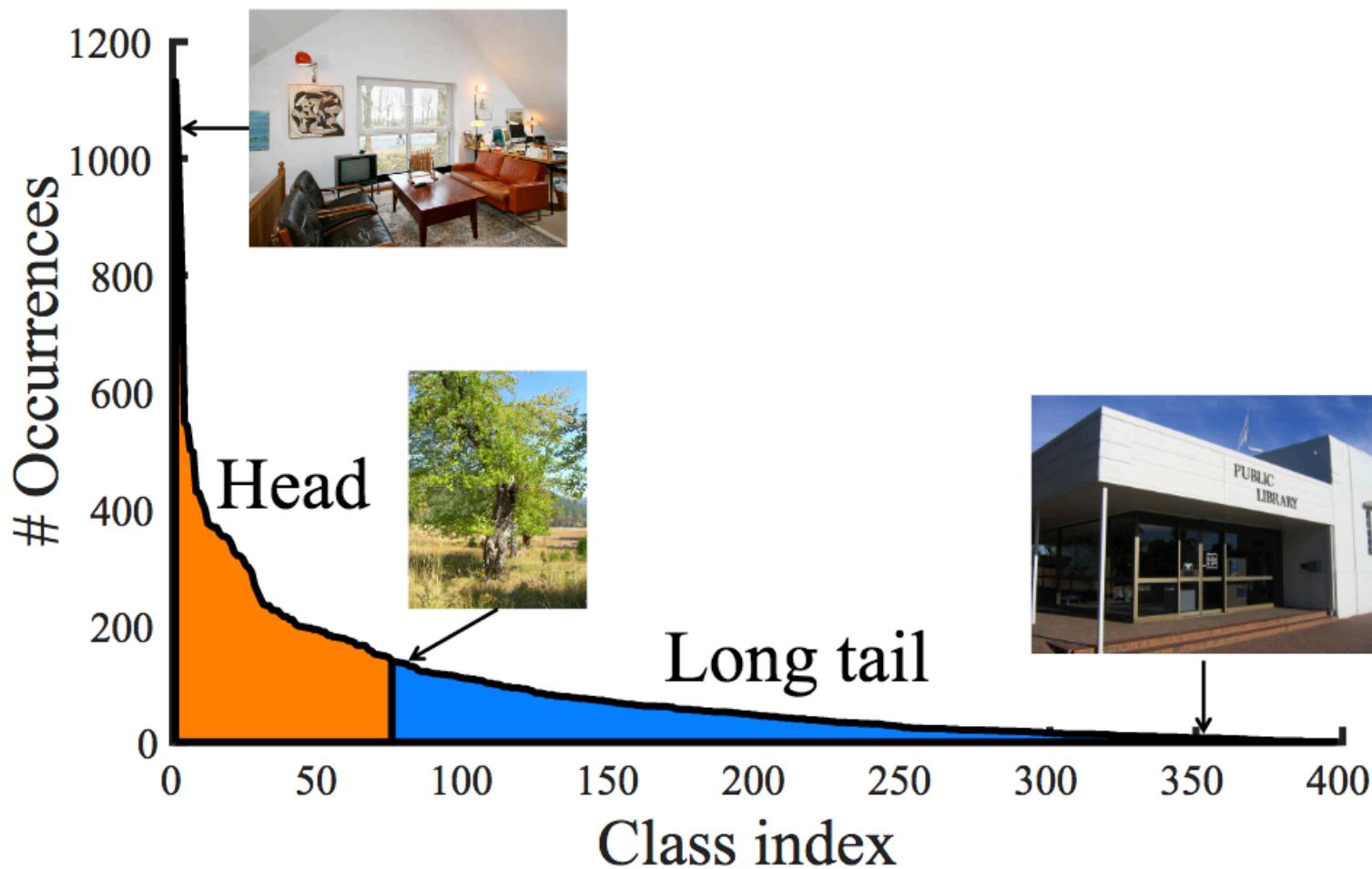
Deep Learning for Computer Vision

Arthur Douillard

Preamble:
Imbalance Learning



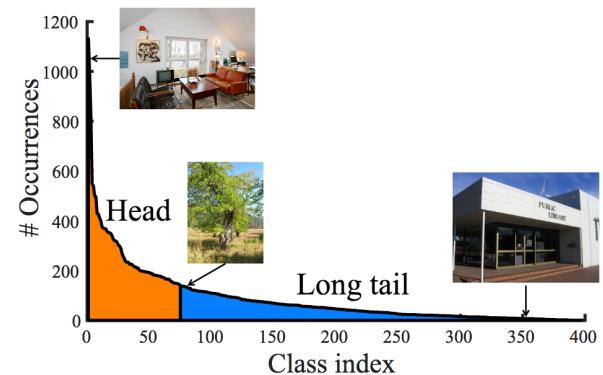
Long-tailed distributions



Common Solution to Imbalanced Learning



- **Oversample** minority classes
 - Works better than undersampling majority classes
- **Class weight** to give more/less importance in the loss to minority/majority classes
 - Before averaging all losses in a batch, multiply each loss by the weight associated to the class ground-truth
- **Sample weight** is like class weight but per sample
 - Example: hard mining
- **Metric Learning** learns a metric not actual classes
 - More on it in the next course!



Multi-labels

Multi-Labels Classification



Classification



“cat”

$[0, 0, 0, 0, \dots, 1, 0, \dots, 0]$

Multi-Labels Classification

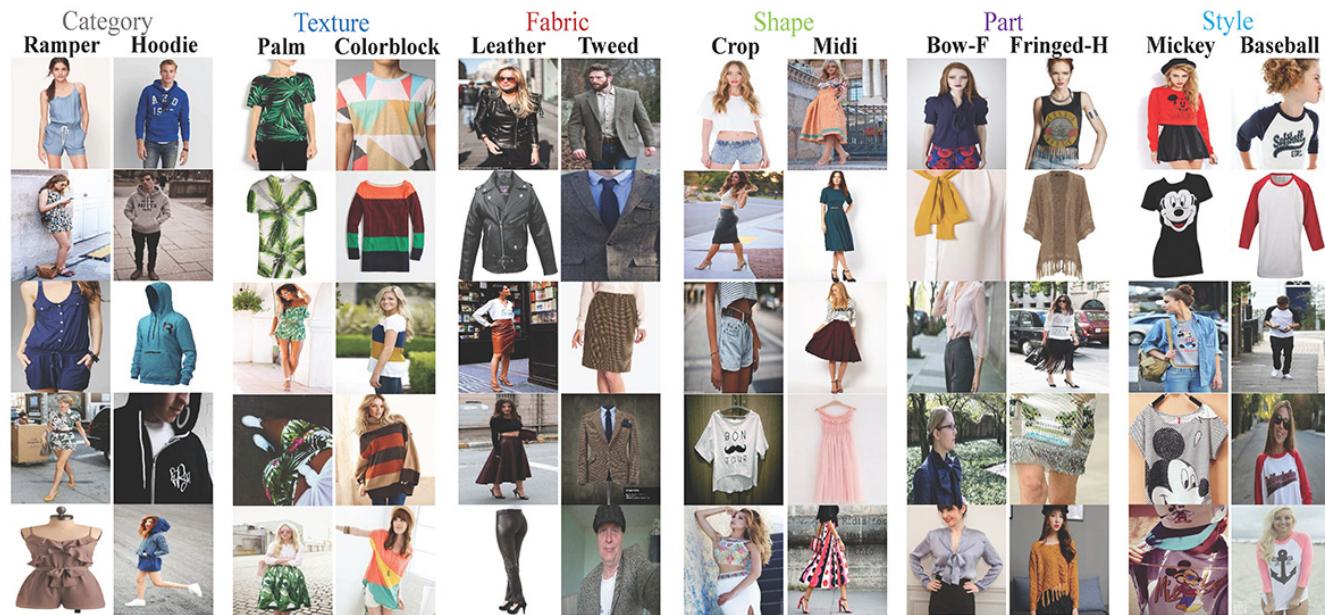


$\{“cat”, “dog”\}$

$[0, 0, 1, 0, \dots, 1, 0, \dots, 0]$



Sparse & Imbalanced

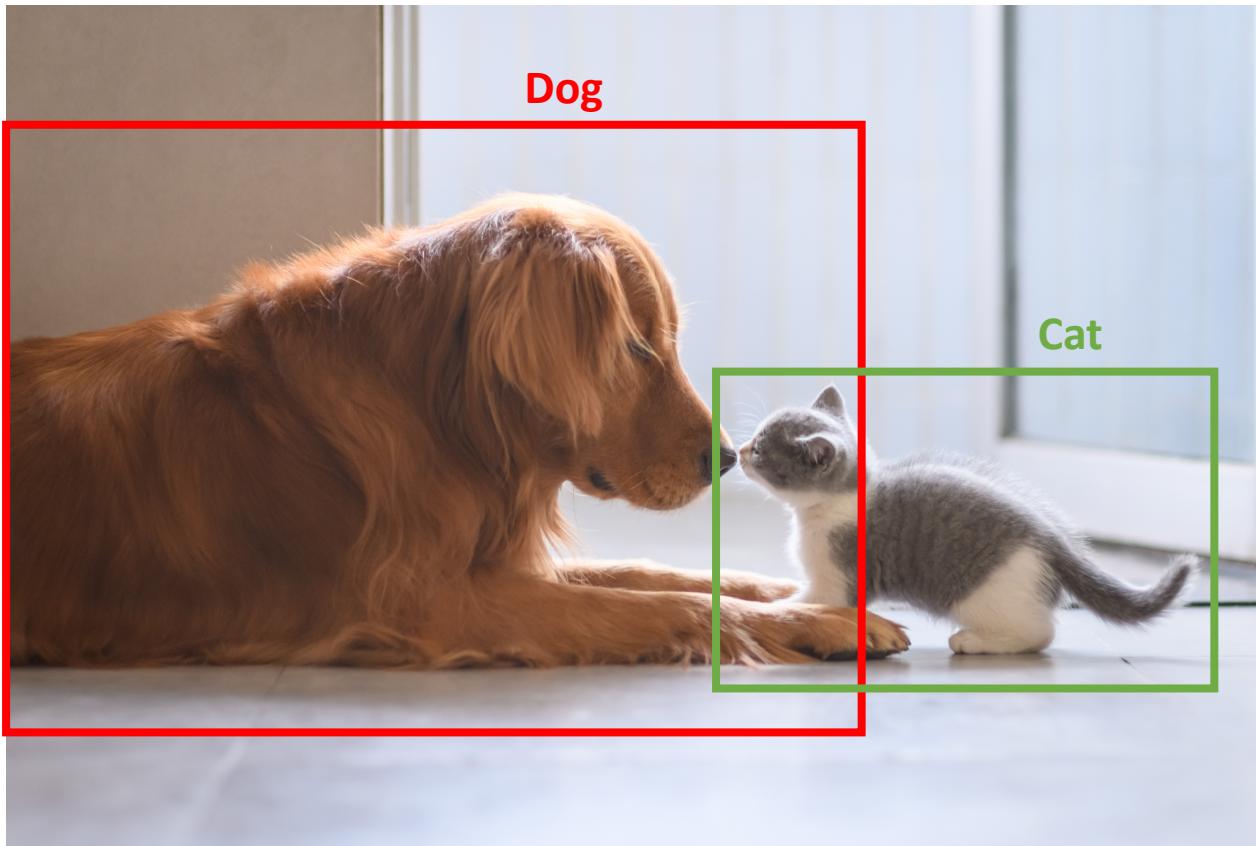


1,000 clothing attributes

Some are very common others not → class weights can be useful

Object Detection

Object Detection

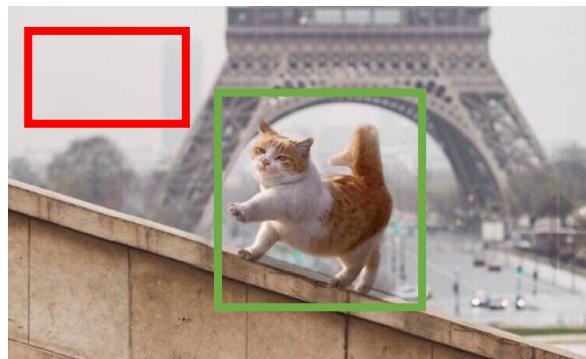


Bounding boxes around the object: (x, y, w, h)
Class prediction: c

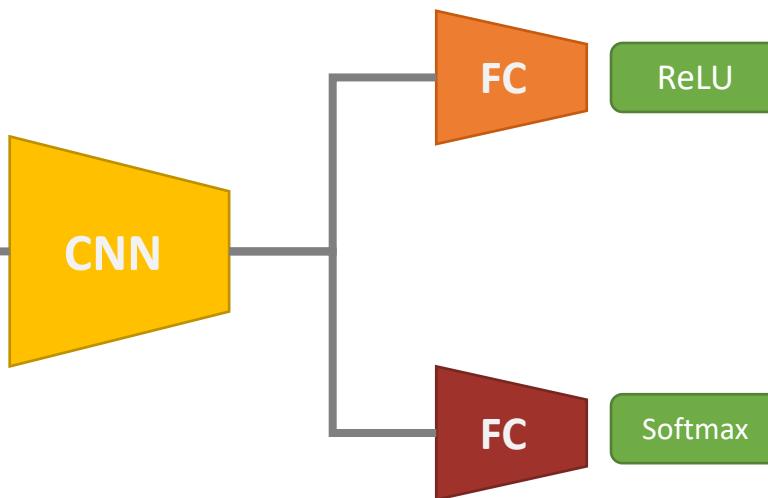
Localizing a single class



Prediction



Ground-truth



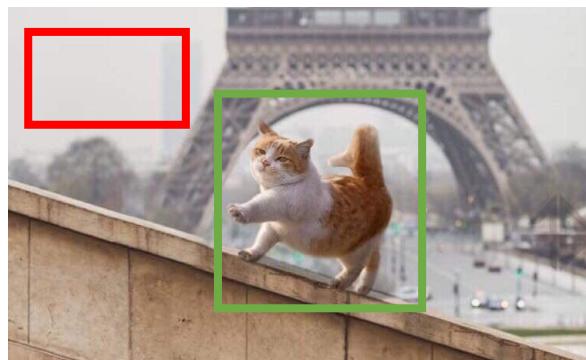
Regression:
 (x, y, w, h)
L2 Loss

Classification:
 (c_1, \dots, c_n)
Cross-Entropy

Localizing a single class



Prediction



Ground-truth

Features maps before global pooling

CNN

FC

ReLU

FC

Softmax

Regression:
 (x, y, w, h)
L2 Loss

Classification:
 (c_1, \dots, c_n)
Cross-Entropy

CNN is pretrained on ImageNet

Localizing multiple classes



Two tasks:

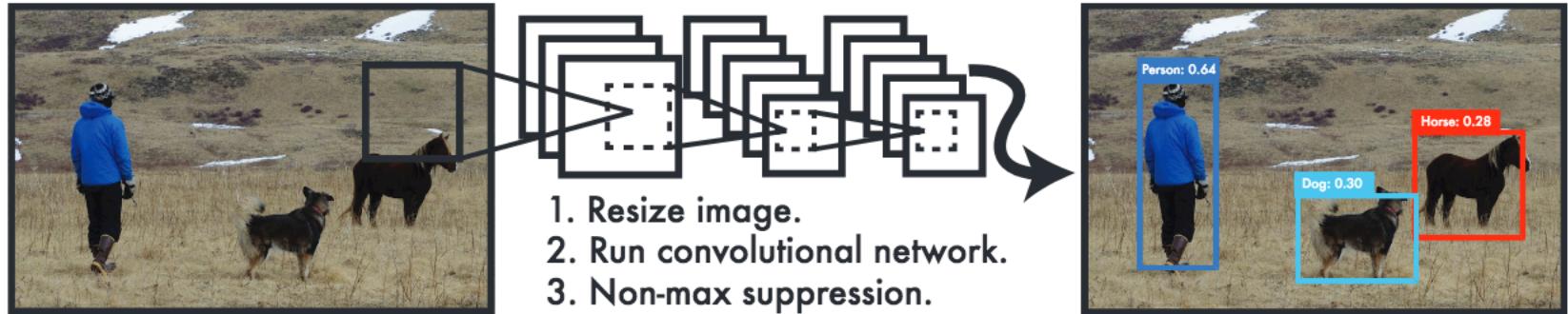
1. Find the **regions of interest** (RoIs) of the image
2. Classify them as either a known class (c_1, c_2, \dots, c_n) or **background** class

Two families:

- **Two-Stage**: first determine which regions may contains an object (1), then classify the said object (2)
→ Fast R-CNN, Faster R-CNN, Mask R-CNN, etc.
- **Single-Stage**: solve the two tasks together
→ SSD, Yolo, RetinaNet, etc.



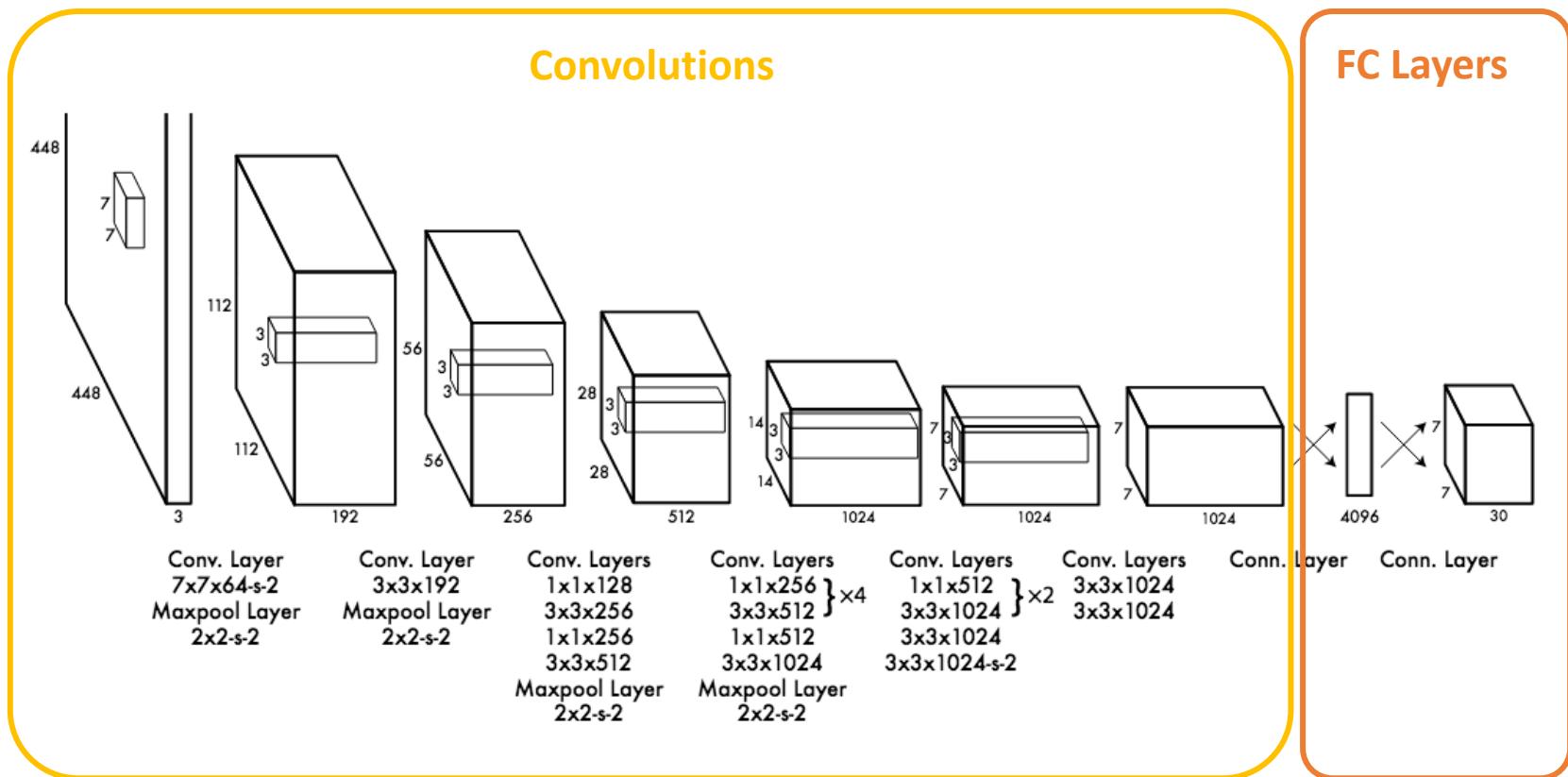
Single Stage: Yolo: You Only Look Once



1. Resize image to 448x448
2. Extract features with [DarkNet](#) and apply object detection procedure
3. Non-max suppression to remove overlapping predictions



Single Stage: Yolo: You Only Look Once



Last layer predict a vector of size 1470:

$$1470 = 7 \times 7 \times (2 \times 5 + 20) = S \times S \times (B \times 5 + C)$$

S : number of cells per side

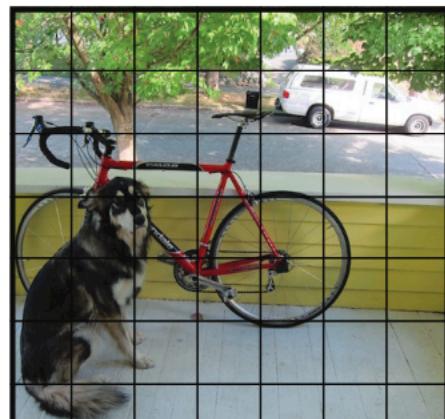
B : number of bounding boxes per cell

C : number of classes (including background class)

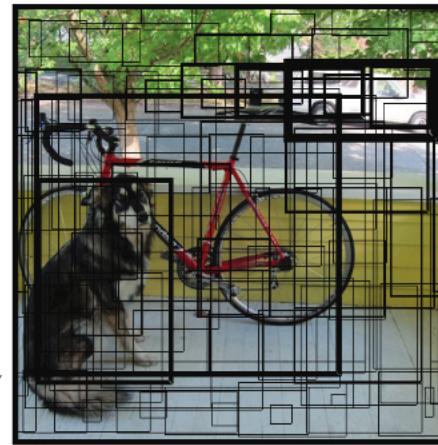


Single Stage: Yolo: You Only Look Once

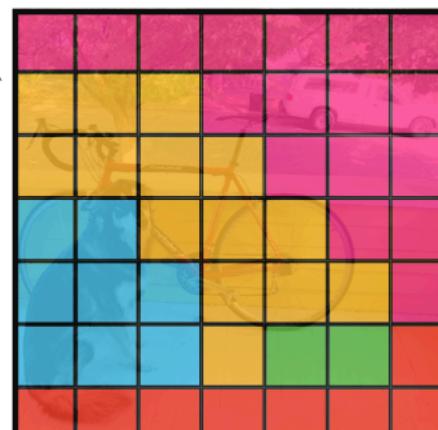
B boxes per cell, and the regression head predicts how to deform them to best fit the object



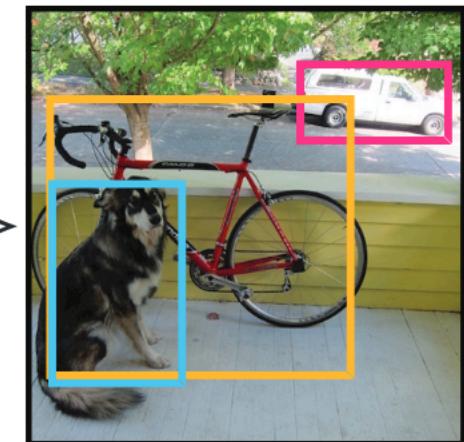
$S \times S$ grid on input



Bounding boxes + confidence



Class probability map



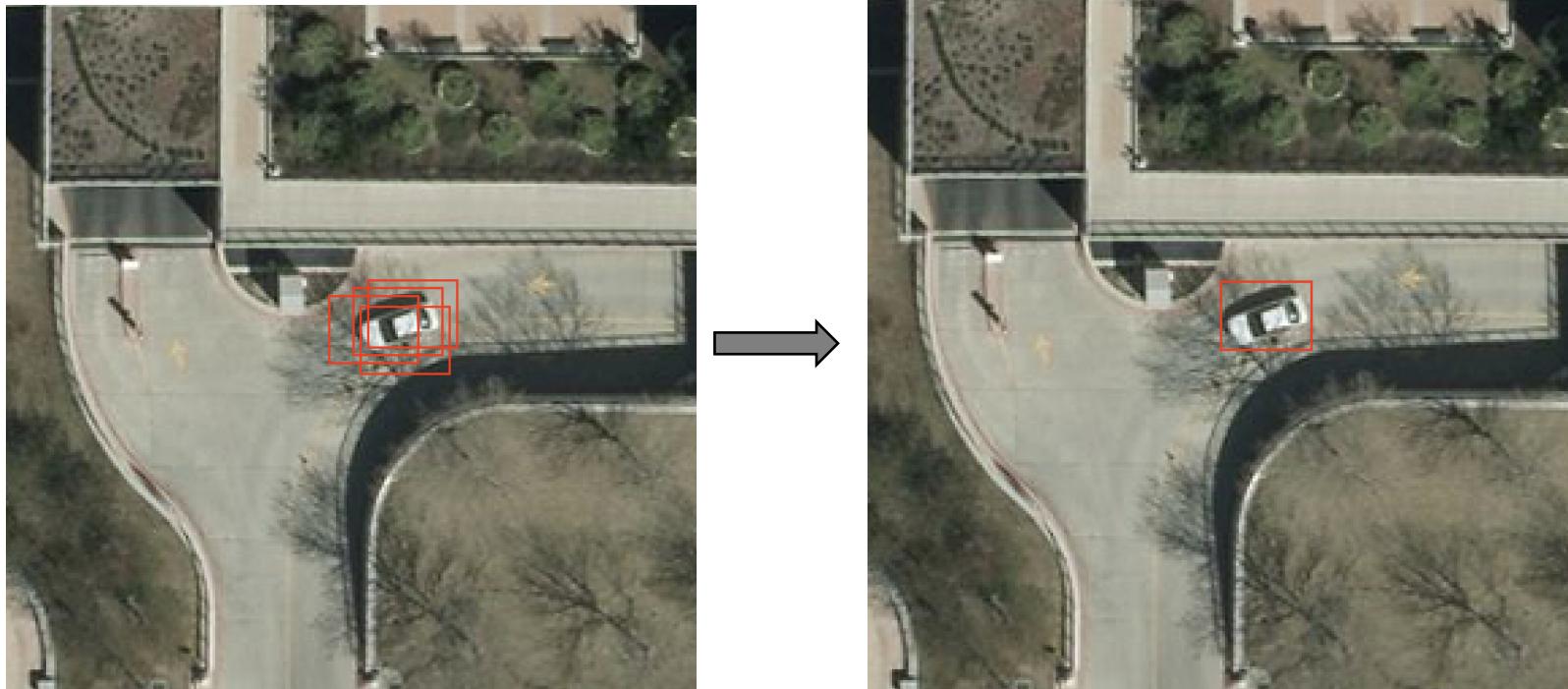
Final detections

How to get only a few good bounding boxes?

Background + NMS



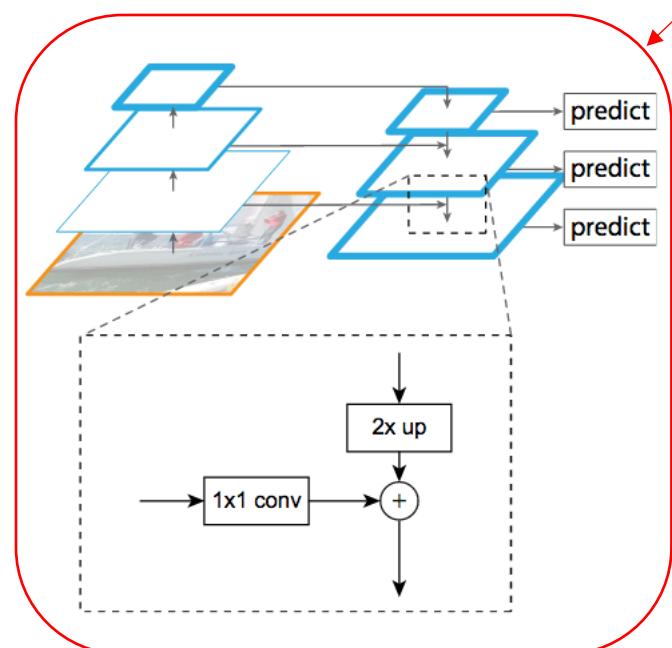
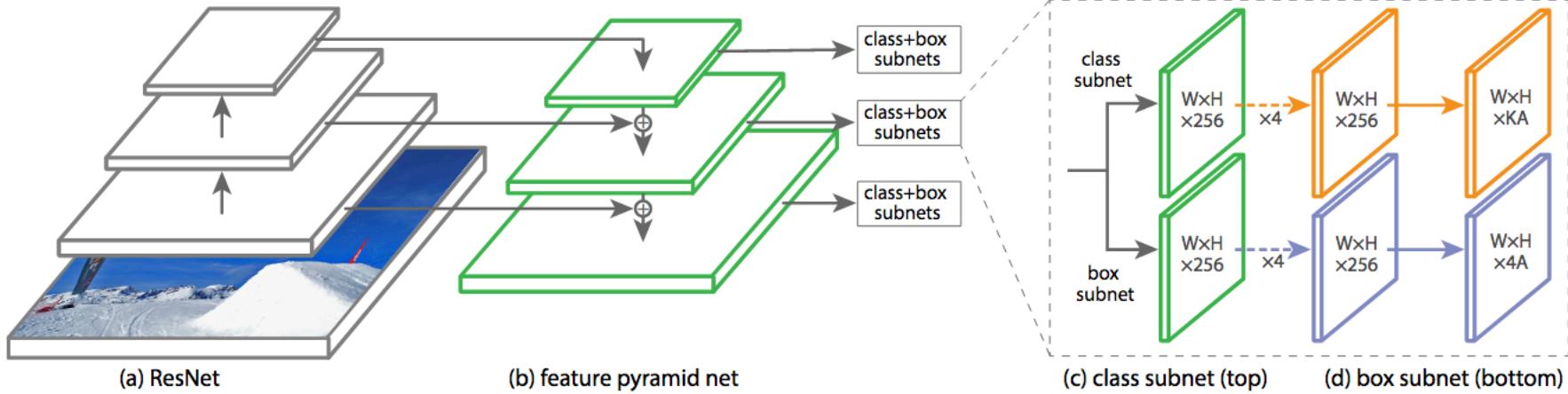
1. Remove all bounding boxes associated to the class “*background*”
2. Then apply **Non-Max Suppression** (NMS)



Among all overlapping bounding boxes, keep the most confident one.

Non differentiable post-processing done in inference!

Single-Stage: RetinaNet



Fully Convolutional Architecture (FCN)

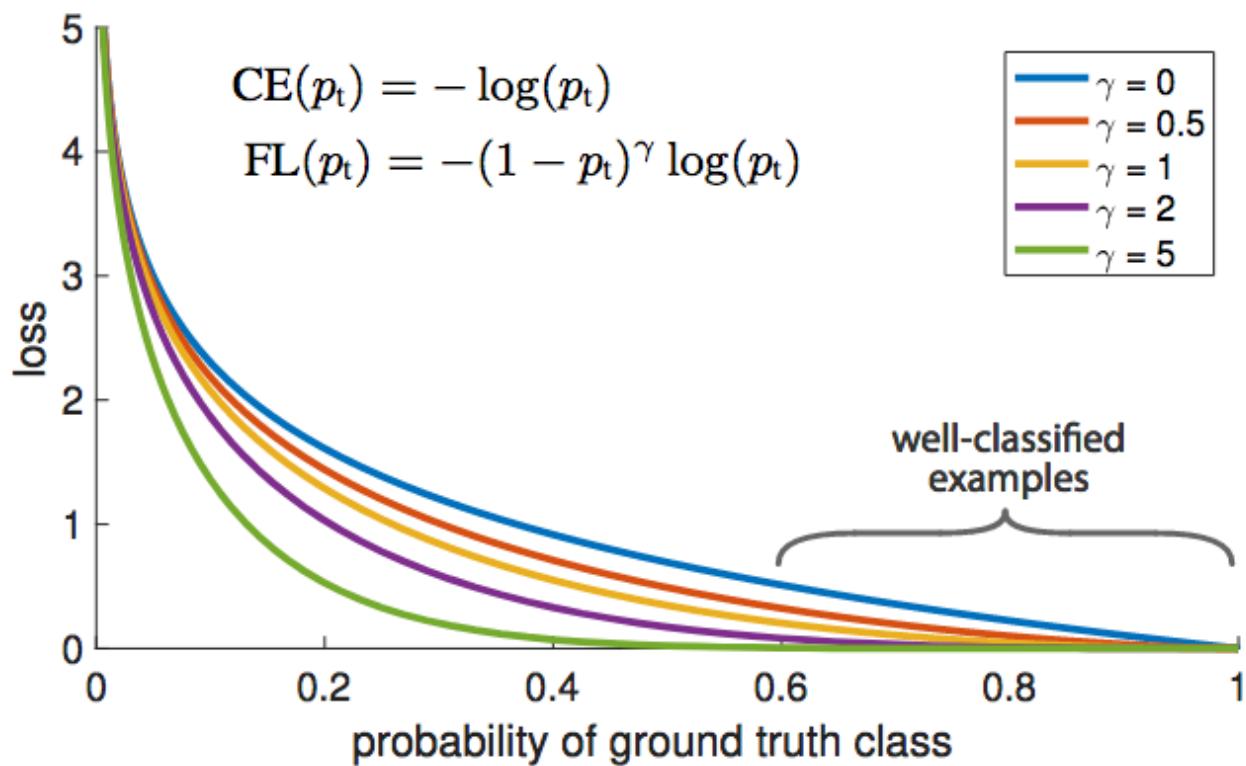


Single-Stage: RetinaNet's Focal Loss

Single-stage models have a huge imbalance with many potential Rols that are **background**.

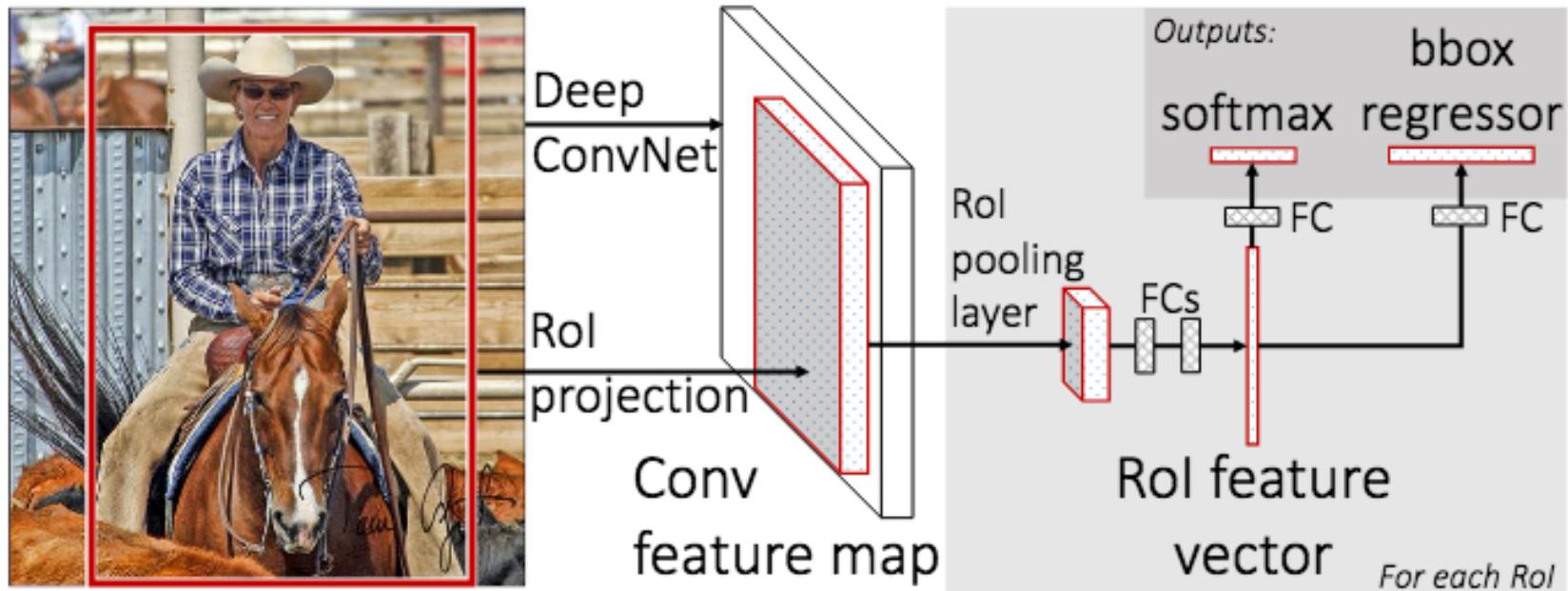
Focal loss reduces the loss if the correct class is predicted with too much confidence.

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$





Two Stages: Fast R-CNN



1. Extract features from pixels with “Deep ConvNet”
2. Generate Rols from pixels using Selective Search
3. Index the features using the Rols



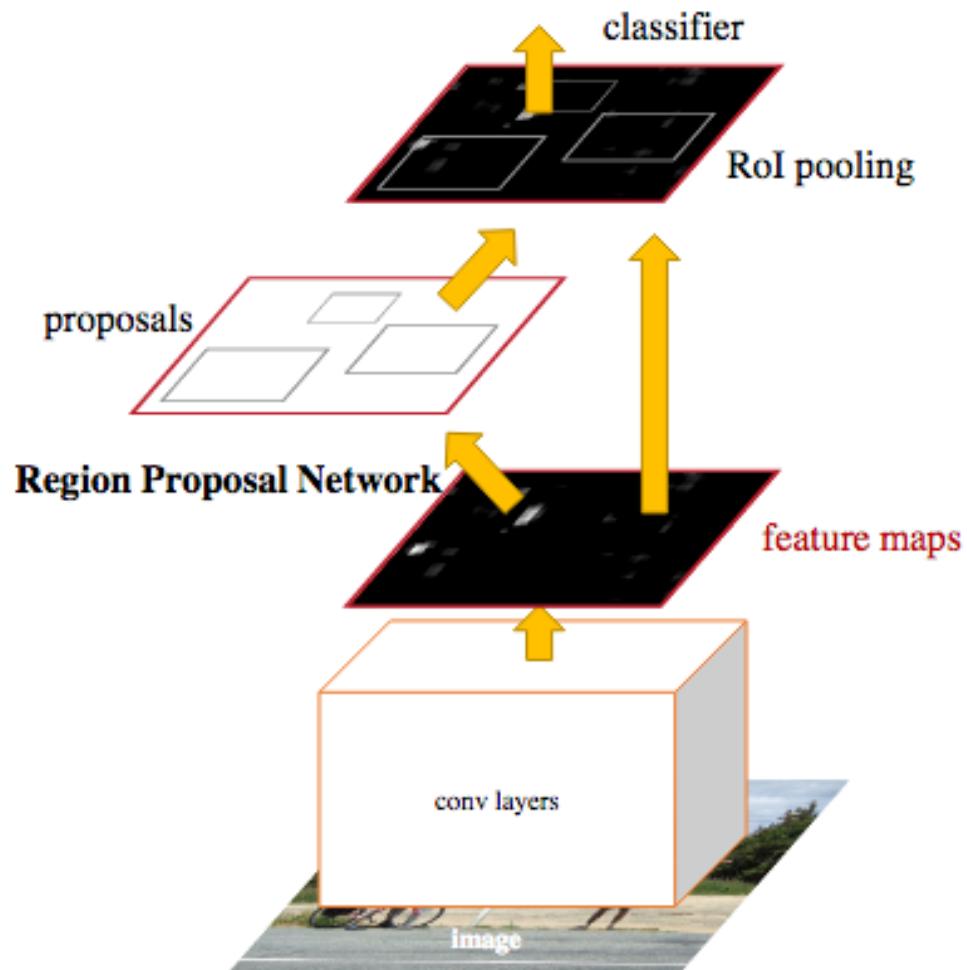
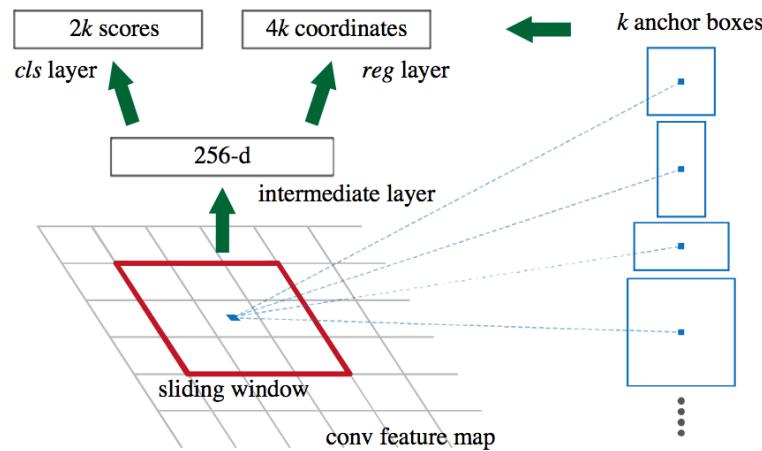


Two Stages: Faster R-CNN

Similar to Fast R-CNN but ditch the selective search for a **Region Proposal Network (RPN)**.

Mini-network that slides different anchors and predict a binary classification:

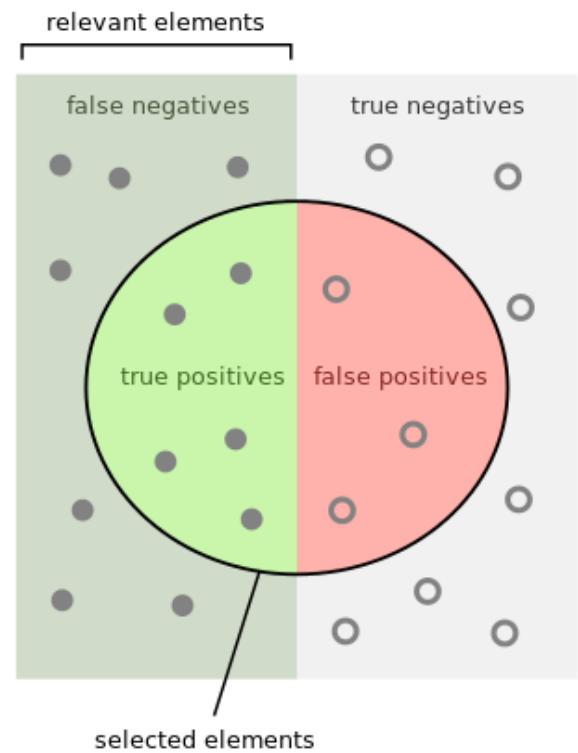
background vs non-background



Metric: Precision & Recall



$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$



How many selected items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{green} + \text{red}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{green}}{\text{green} + \text{light green}}$$

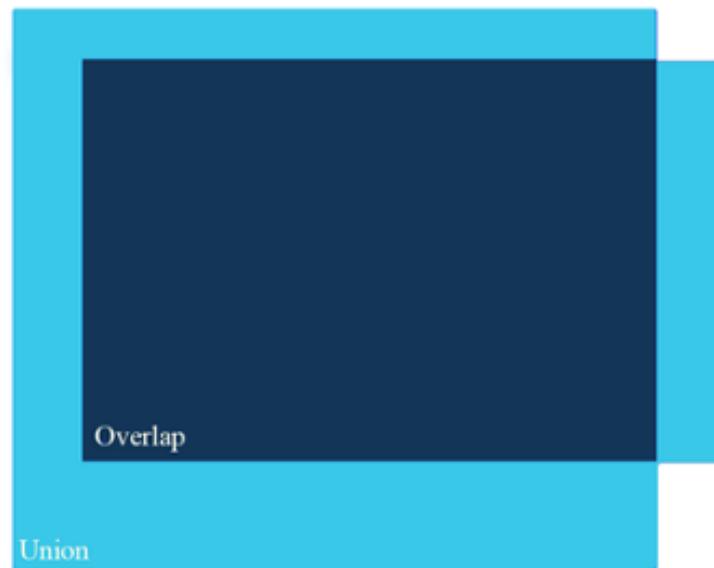


Metric: Intersection over Union

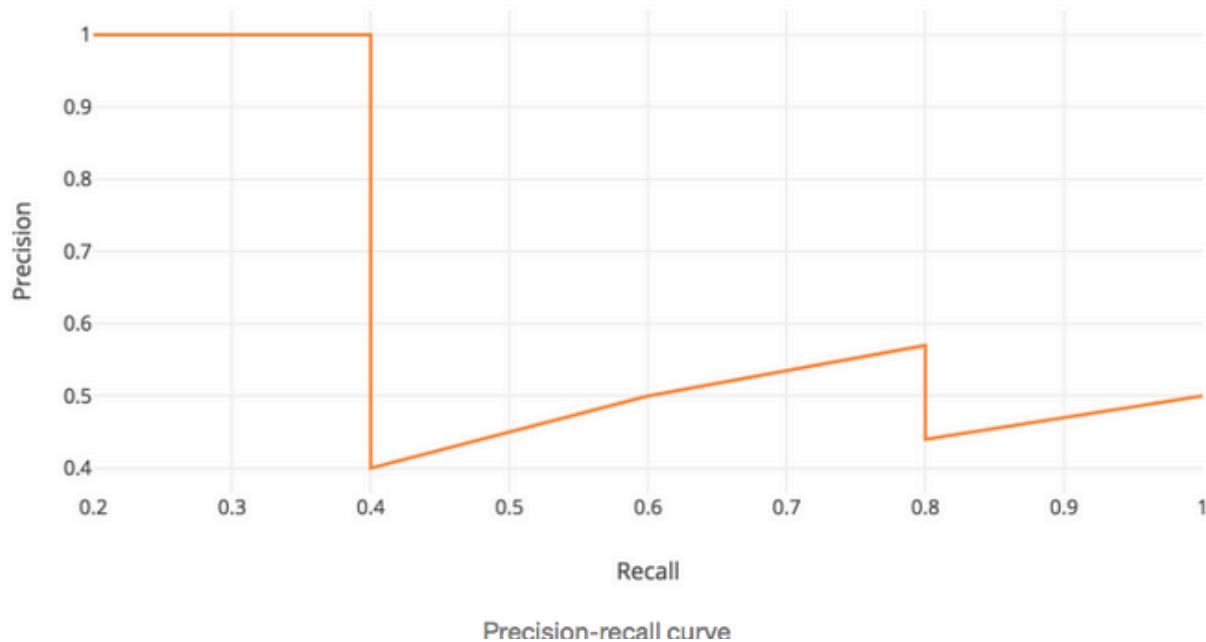


- Ground truth
- Prediction

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$



Metric: mean Average Precision (mAP)



$$AP = \int_0^1 p(r)dr$$

An object is correctly predicted if $\text{IoU} > 0.5$ with ground-truth.

AP is the area under the curve for the graph Recall-Precision.

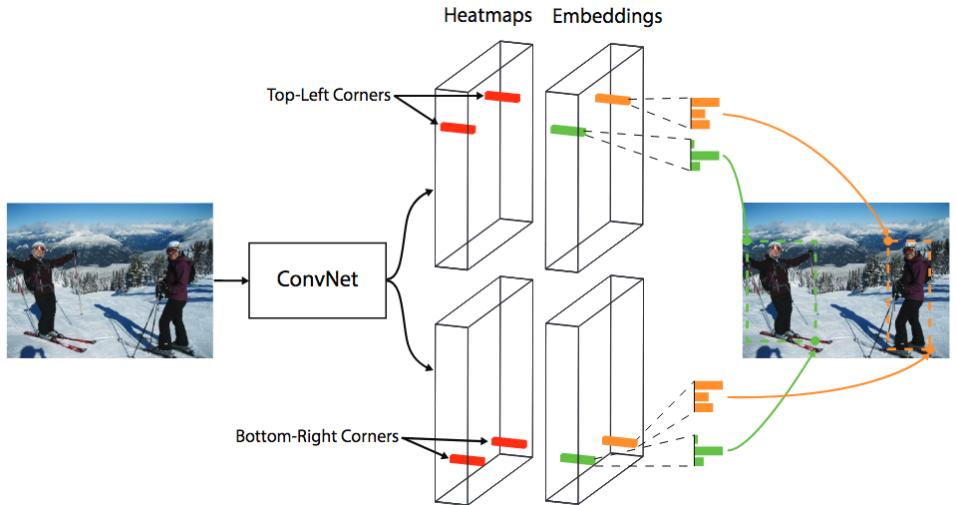
→ We want a high precision while not missing an object (recall)

Beyond anchors



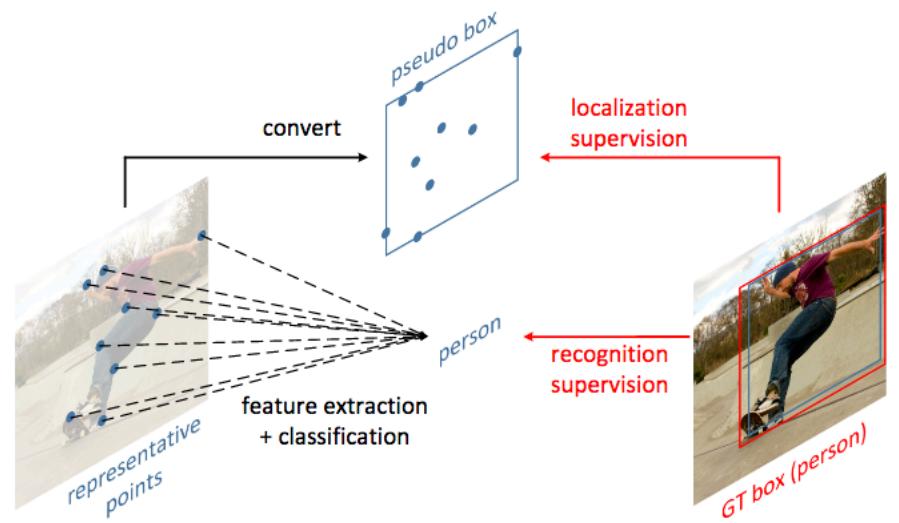
CornerNet, predicting the top-left and bottom-right corners.

[[Law et al. ECCV 2018](#)]



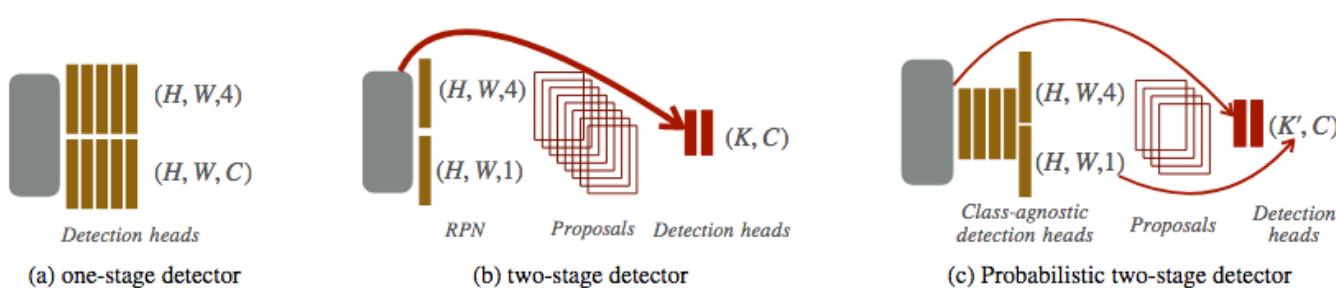
RepPoints, predicting points that bound an object.

[[Yang et al. ICCV 2019](#)]

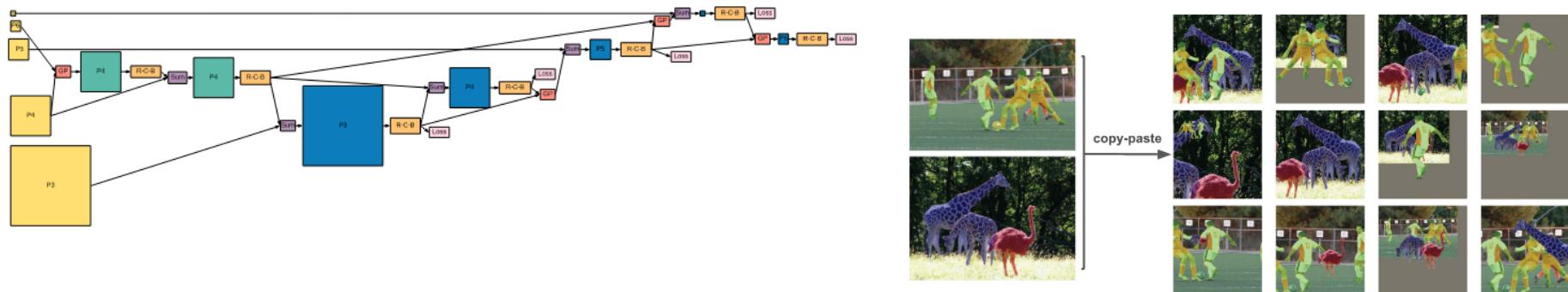




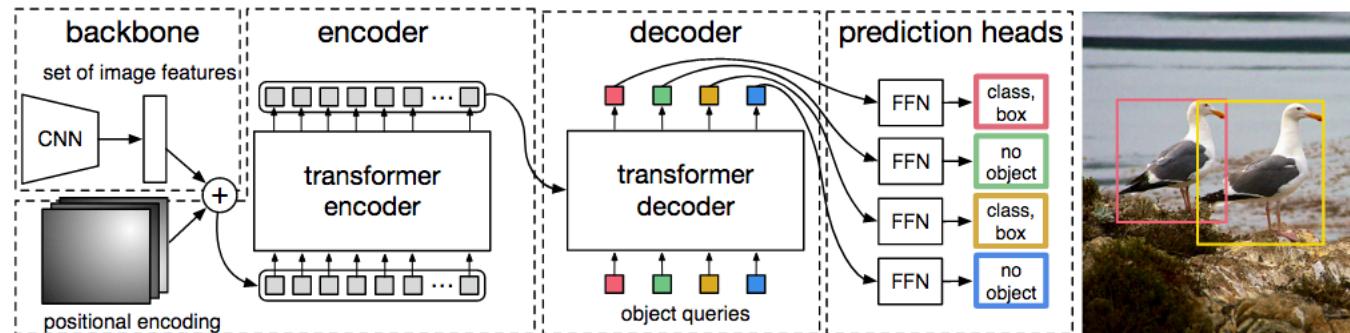
State of the Art



Combining RPN prediction with final class prediction: [[CenterNet2, Zhou et al. arXiv 2021](#)]



NAS-FPN + Aggressive copy-pasted based augmentation: [[Ghiasi et al. CVPR 2021](#)]



Transformers-based to produce a set of boxes: [[Carion et al. ECCV 2020](#)]

Segmentation



Semantic Segmentation

Each pixel has a class





Instance Segmentation

Each pixel has a foreground class and an instance



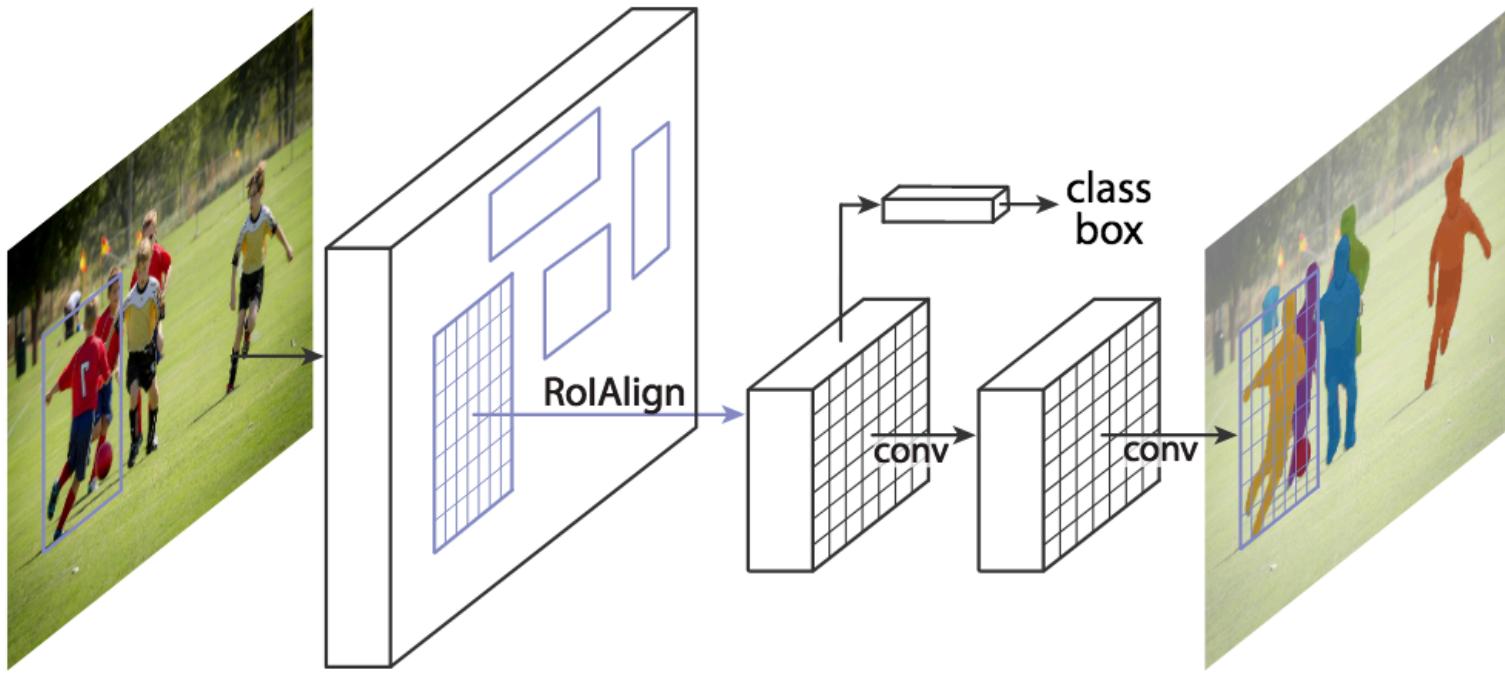
Panoptic Segmentation



Semantic + Instance Segmentation

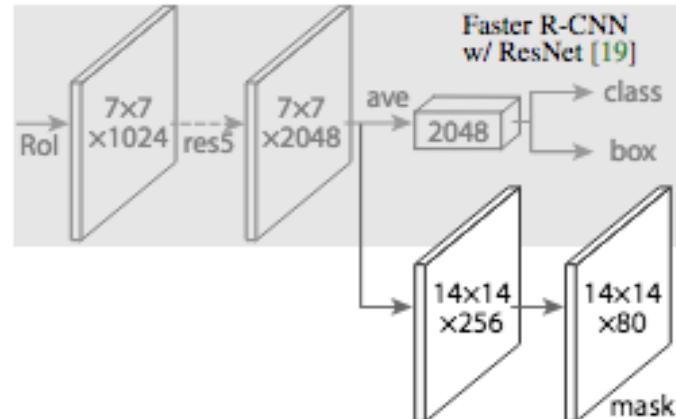


Instance Segmentation: Mask R-CNN

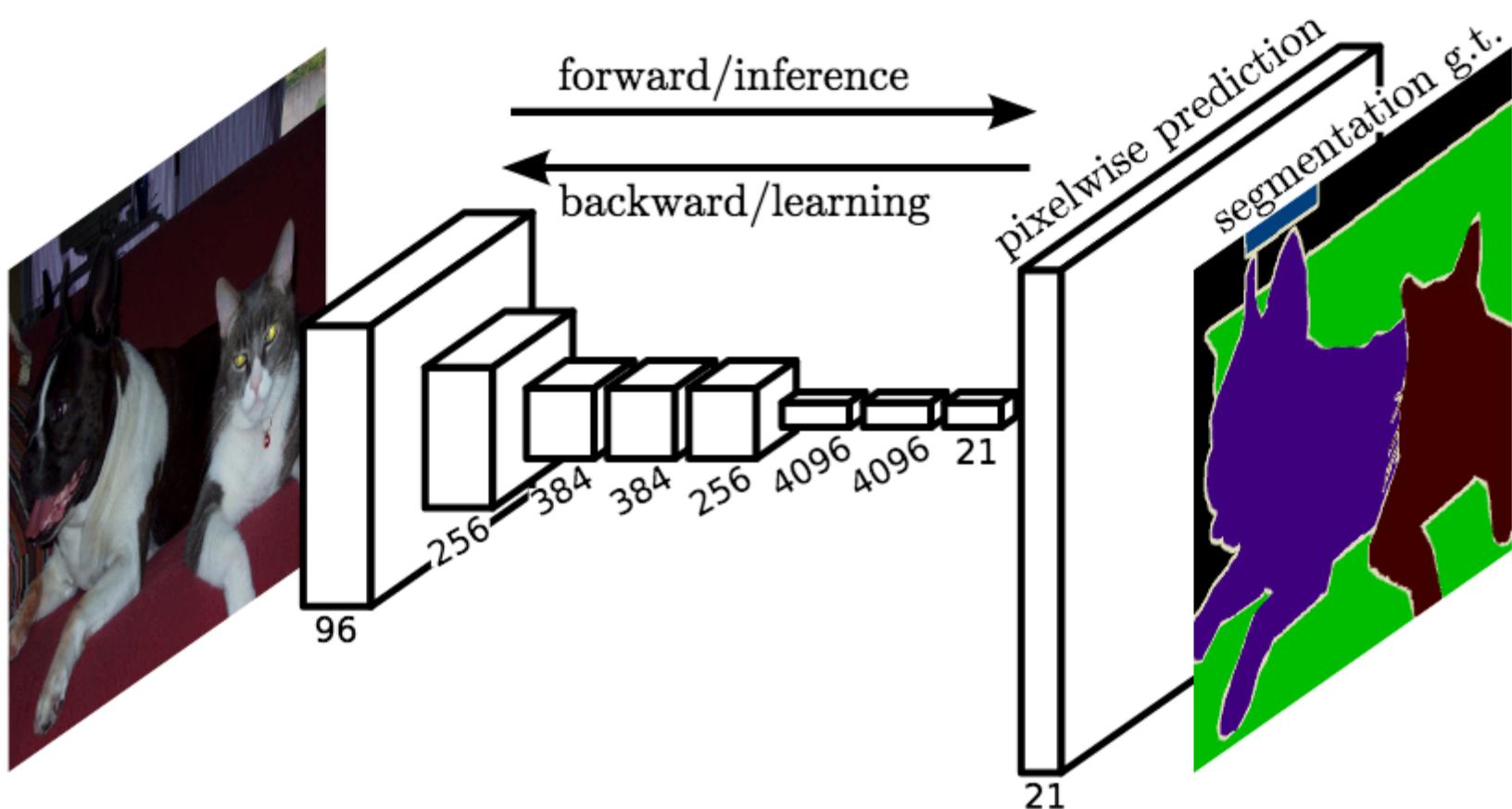


Extend Faster R-CNN with a new branch made only of convolutions.

The branch predict a class per pixels per box.

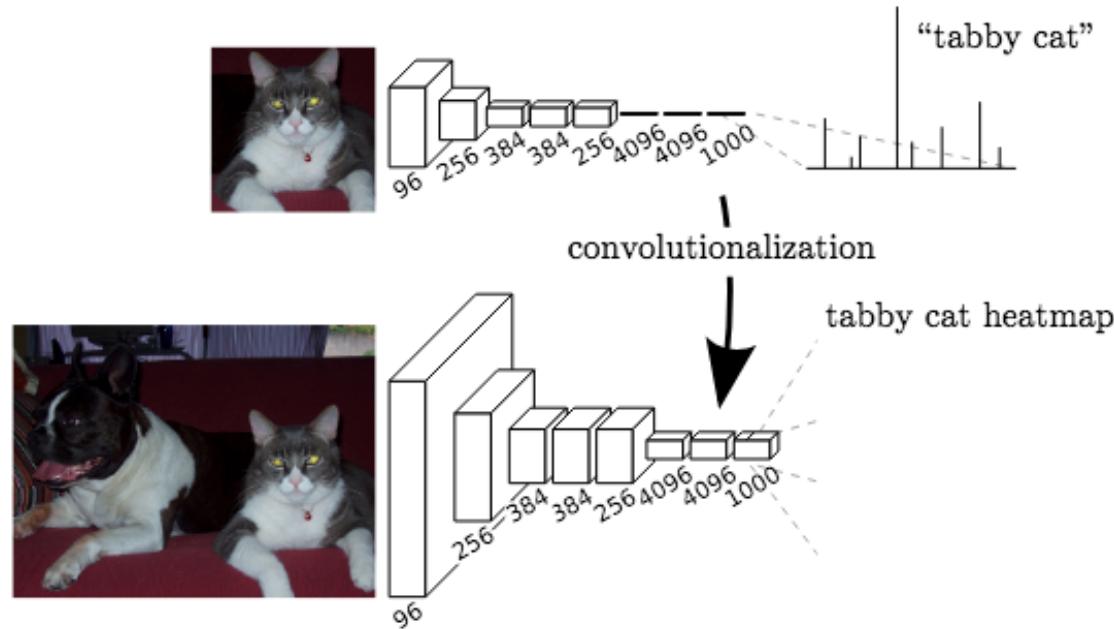


Semantic Segmentation: Fully Convolutional Network



Predict 21 classes per features/logits pixels, upscale the prediction to image size and classify per pixel with cross-entropy.

Semantic Segmentation: Fully Convolutional Network



Init the network with model pretrained on ImageNet.

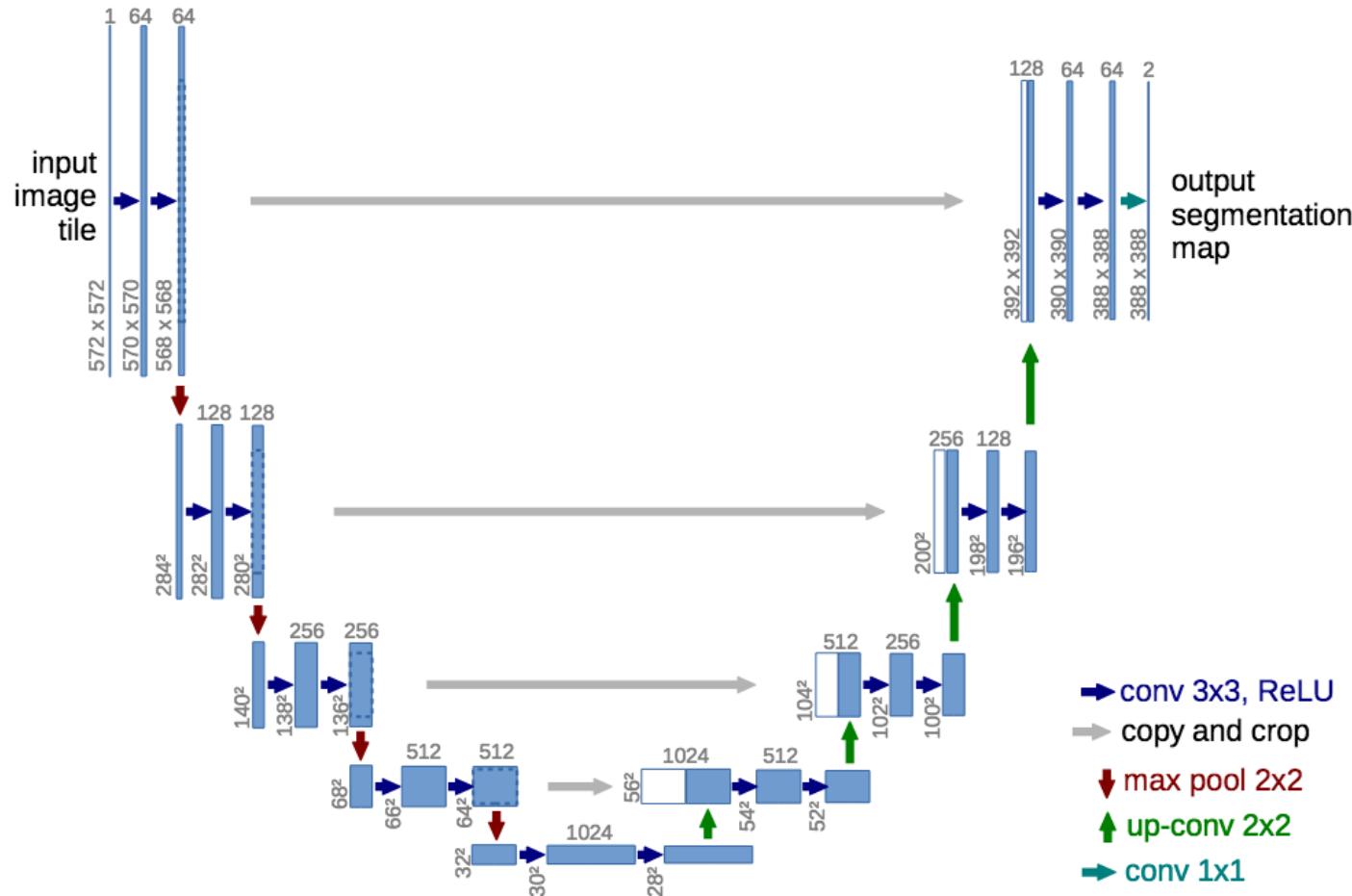
And convert the last FCs layers into 1x1 convolutions... while keeping weights!

FC weight: $C_i \times C_o$

Conv weight: $C_i \times C_o \times k \times k$



Semantic Segmentation: U-Net

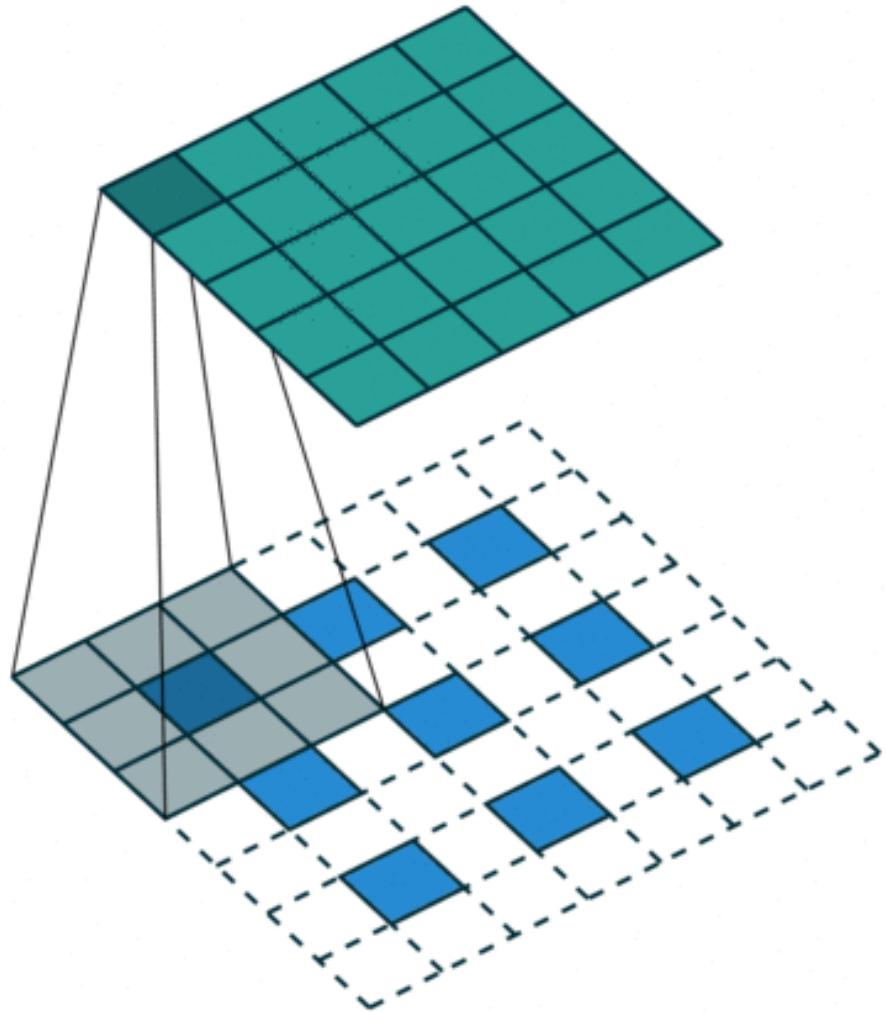


Upsample the small features maps with **transposed convolution**

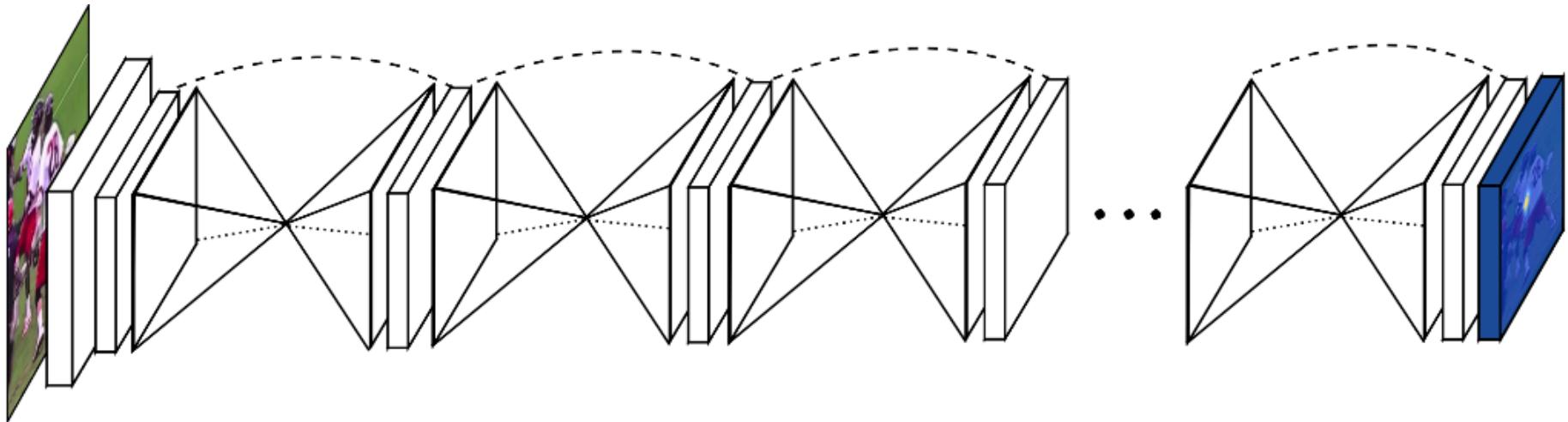
Transposed Convolution



Add padding inside the input features.



Semantic Segmentation: Hourglass Network



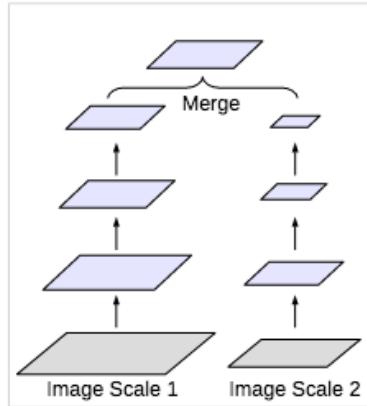
Stack of Hourglass, which are similar to U-Net

Each hourglass is trained to predict the final segmentation map.

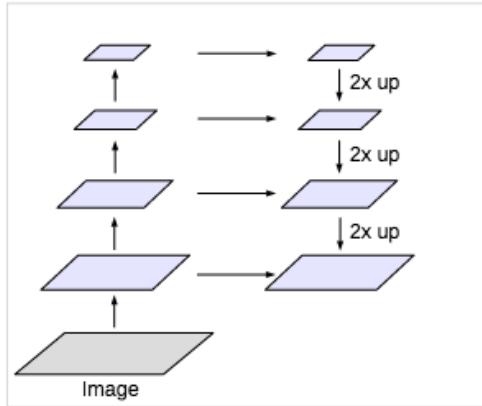
The $N+1$ hourglass refines the results of the N hourglass, as in boosting.



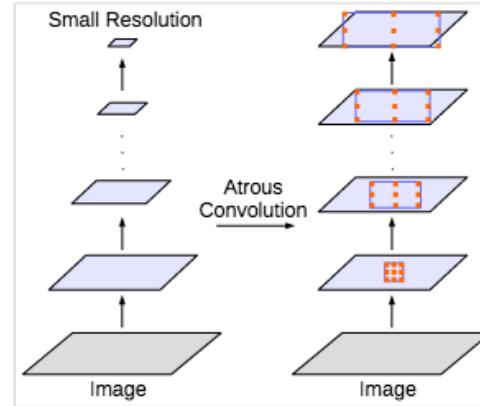
Semantic Segmentation: DeepLab v2-v3



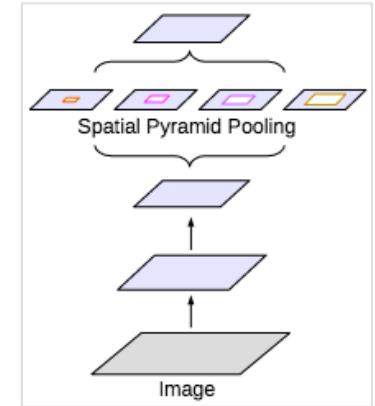
(a) Image Pyramid



(b) Encoder-Decoder



(c) Deeper w. Atrous Convolution



(d) Spatial Pyramid Pooling

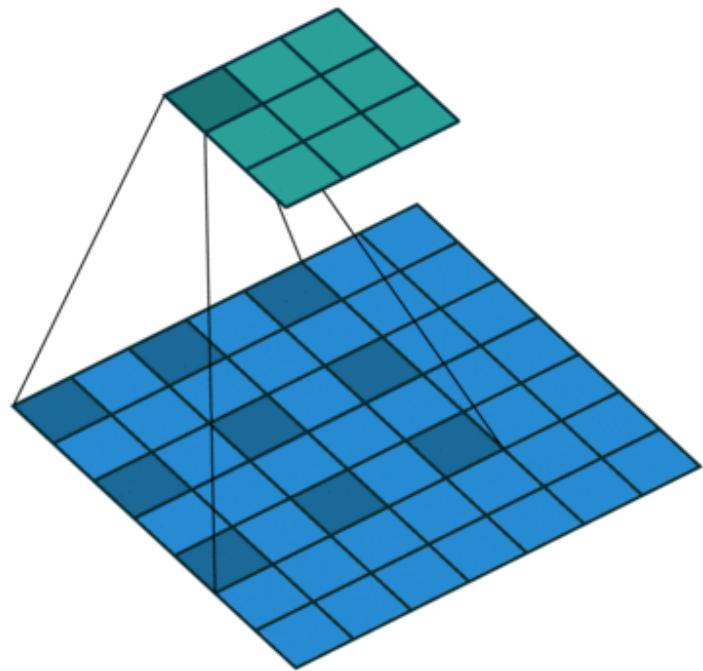
Multi-Scale
images

U-Net like
architectures

DeepLab v2

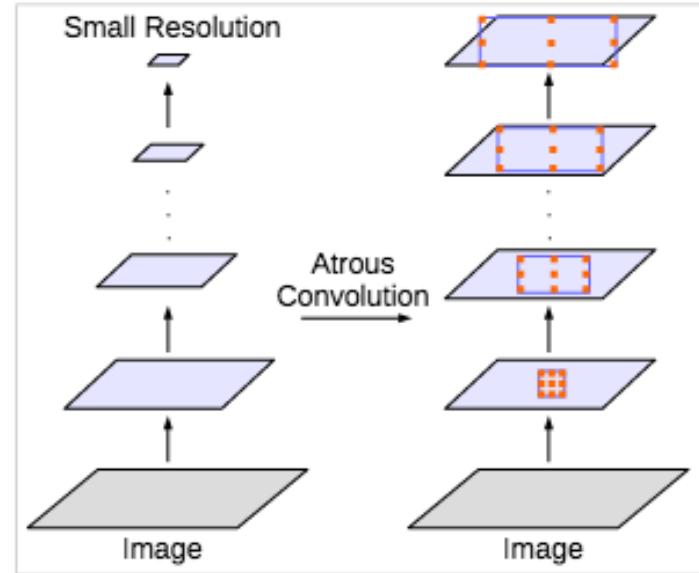
DeepLab v3

Semantic Segmentation: DeepLab v2



Dilated (Atrous) convolutions.

Add padding inside kernel



(c) Deeper w. Atrous Convolution

- **Larger receptive field-of-view**
- Add also a lot of padding around the input features to keep large resolution

Semantic Segmentation: DeepLab v3



Atrous Spatial Pyramid Pooling (ASPP):

→ Combine multiple dilation rate, each seeing a different “scale”

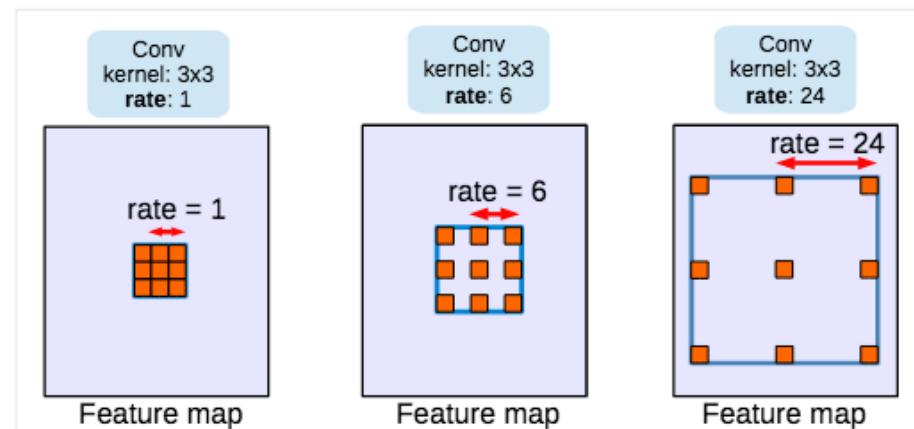
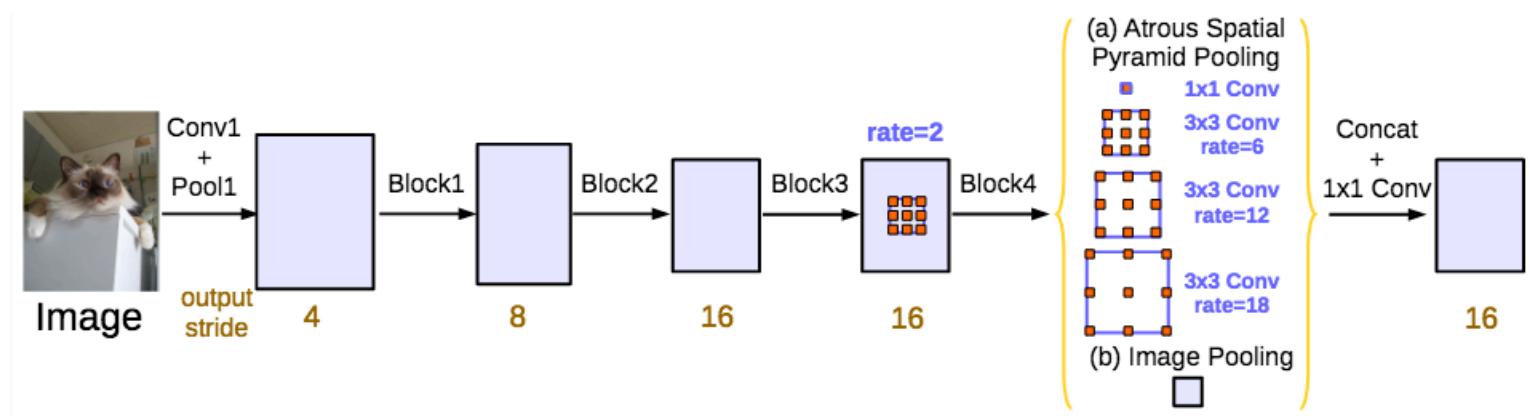


Figure 1. Atrous convolution with kernel size 3×3 and different rates. Standard convolution corresponds to atrous convolution with $rate = 1$. Employing large value of atrous rate enlarges the model’s field-of-view, enabling object encoding at multiple scales.



Semantic Segmentation: DeepLab v3



Atrous Spatial Pyramid Pooling (ASPP):

→ Combine multiple dilation rate, each seeing a different “scale”

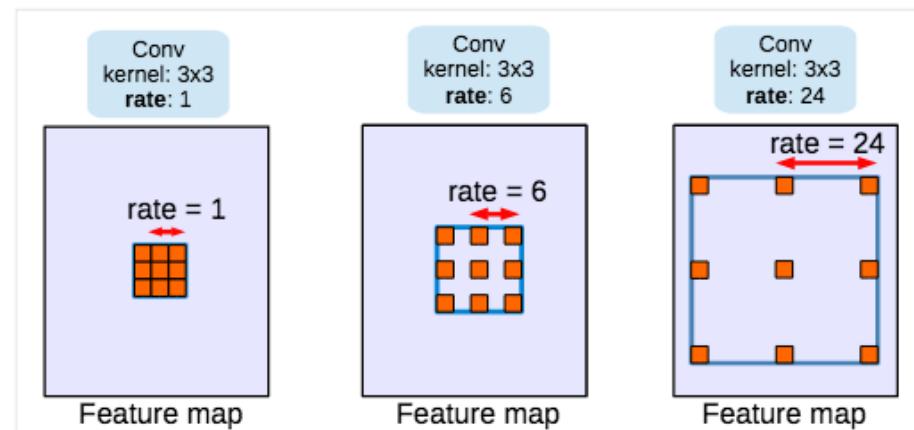
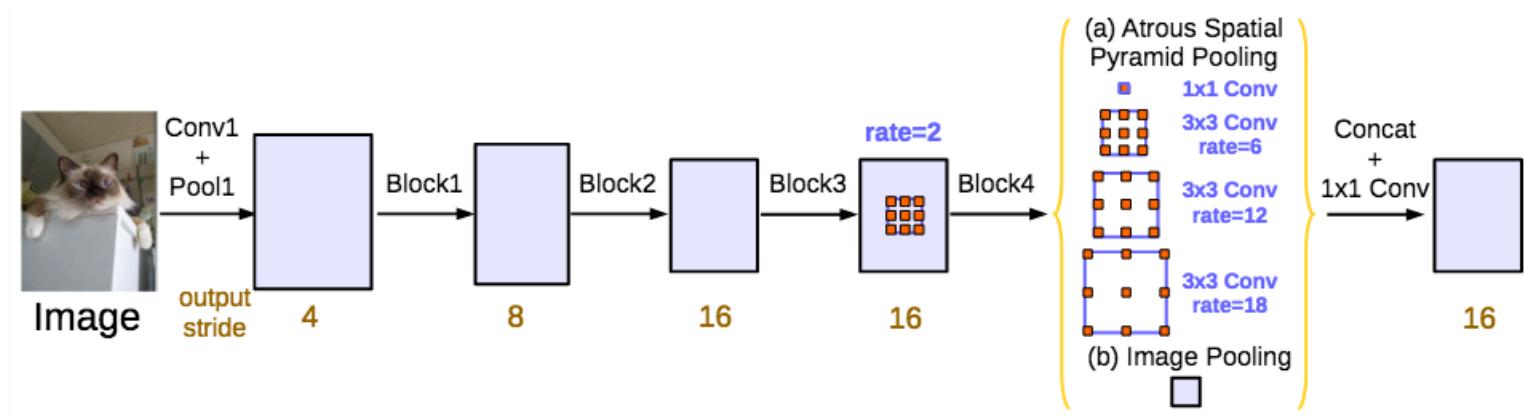


Figure 1. Atrous convolution with kernel size 3×3 and different rates. Standard convolution corresponds to atrous convolution with $rate = 1$. Employing large value of atrous rate enlarges the model’s field-of-view, enabling object encoding at multiple scales.



Two key ideas for segmentation

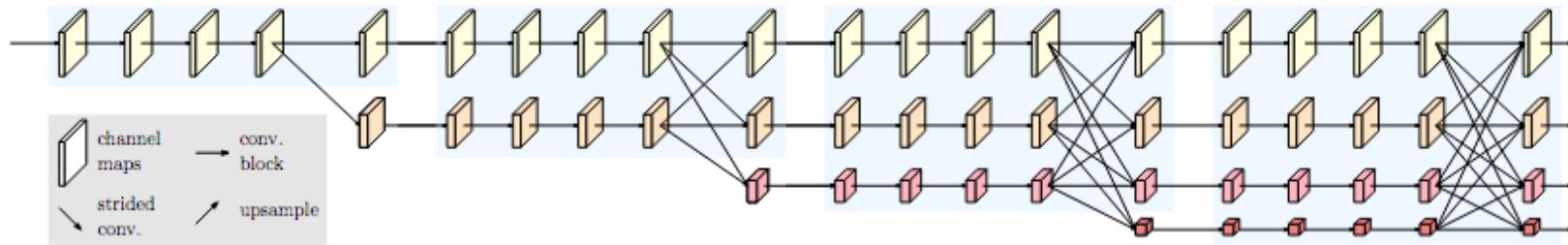
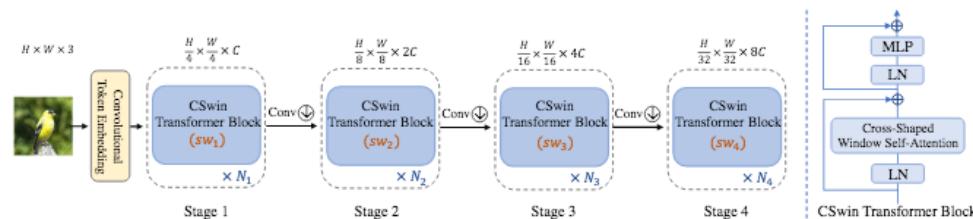
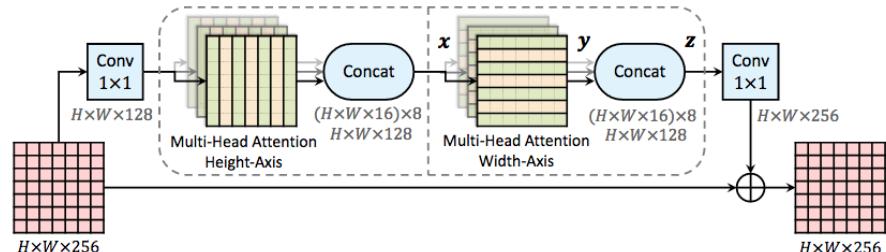


Figure 1. A simple example of a high-resolution network. There are four stages. The 1st stage consists of high-resolution convolutions. The 2nd (3rd, 4th) stage repeats two-resolution (three-resolution, four-resolution) blocks. The detail is given in Section 3.

Multi-scale/resolution is very important to detect all kinds of object. [[Sun et al. CVPR 2019](#)]



Attention to far away pixels is also important.
[[Wang et al. ECCV 2020](#)]

Thus, transformers will probably be a good choice.
[[Dong et al. arXiv 2021](#)]

Given fixed method, two possible improvements



Better backbone:

- Larger, wider, deeper, with all modern tricks (EffNet vs ResNet vs VGG)
- Pretrained on a dataset as large as possible (ImageNet, JFT300M, Instagram1B)

Better augmentation and regularization:

- Many data augmentation are possible (erasing, copy-pasting, etc.)
- Regularization trick may help (which kind of normalization? DropPath? Focal Loss?)

Small break,
then coding session!