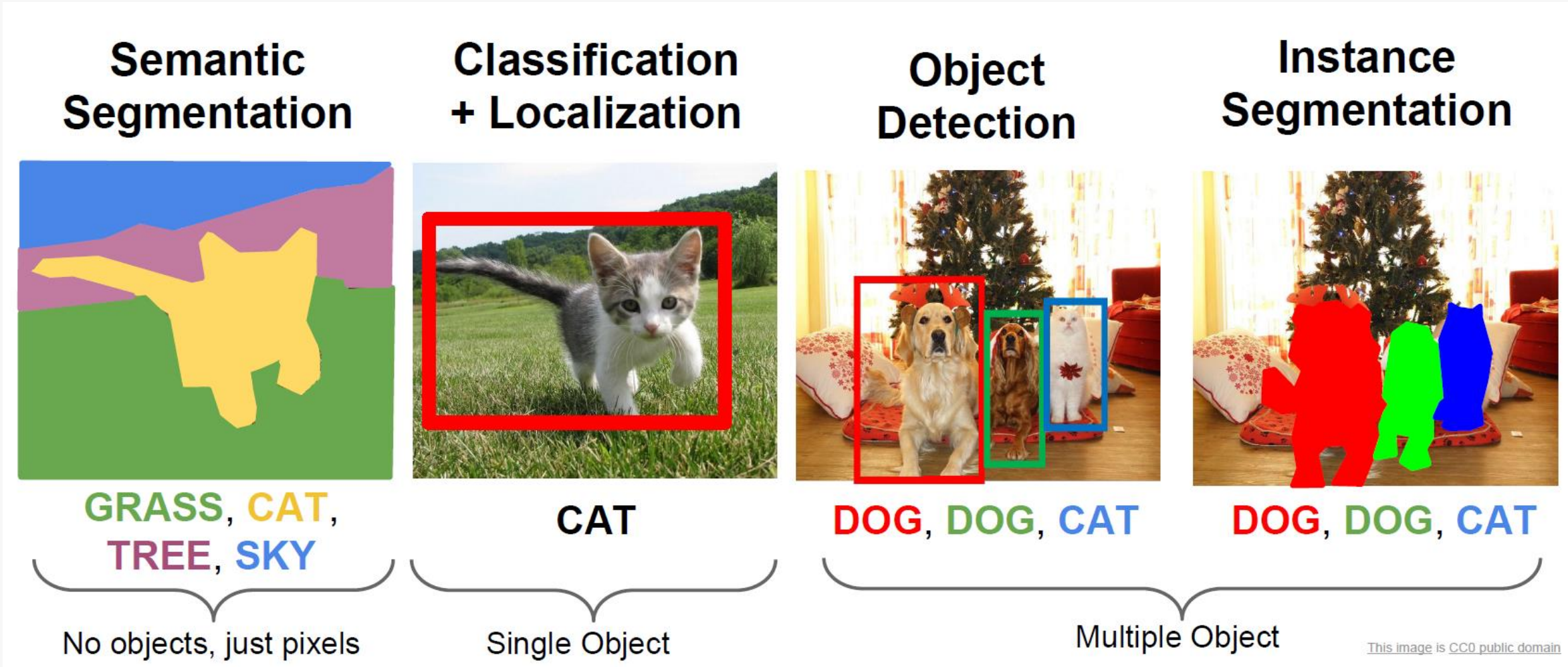
Deep Learning for Computer Vision

Ahmed Hosny Abdel-Gawad

Senior AI/CV Engineer

Object Detection

Computer Vision Tasks

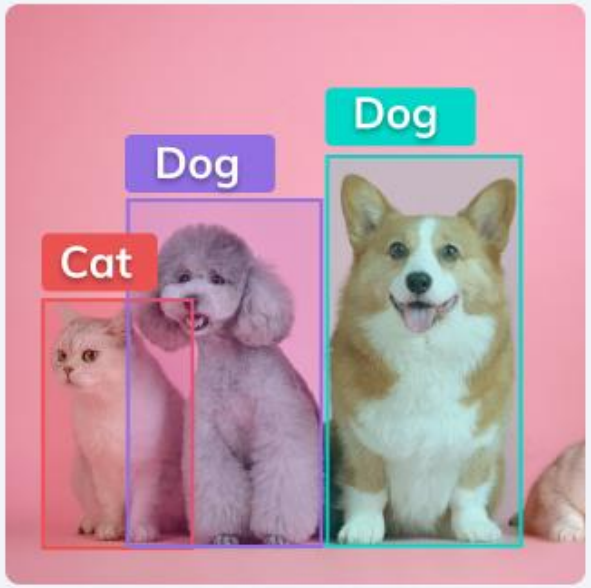


Object Detection vs. Image Classification

Classification

Cat

Detection

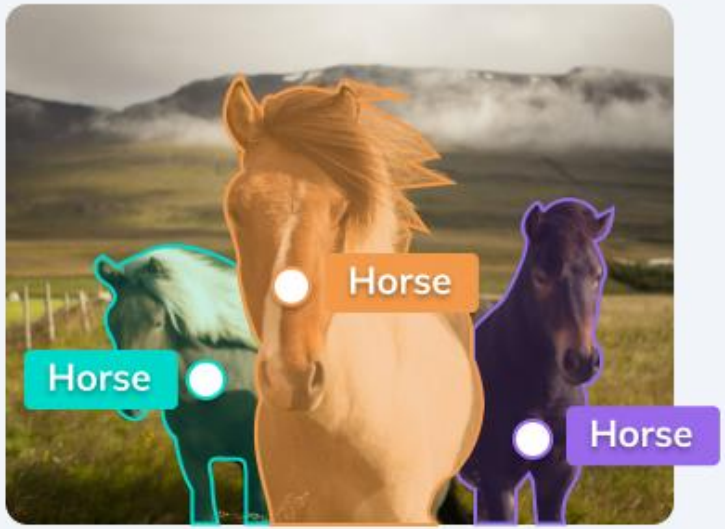


Cat, Dog, Dog

Object Detection vs Image Segmentation

Object detection

Semantic Segmentation



Instance Segmentation

Object Detection + Semantic Segmentation = **Instance Segmentation**

Object Detection: Understanding the Task

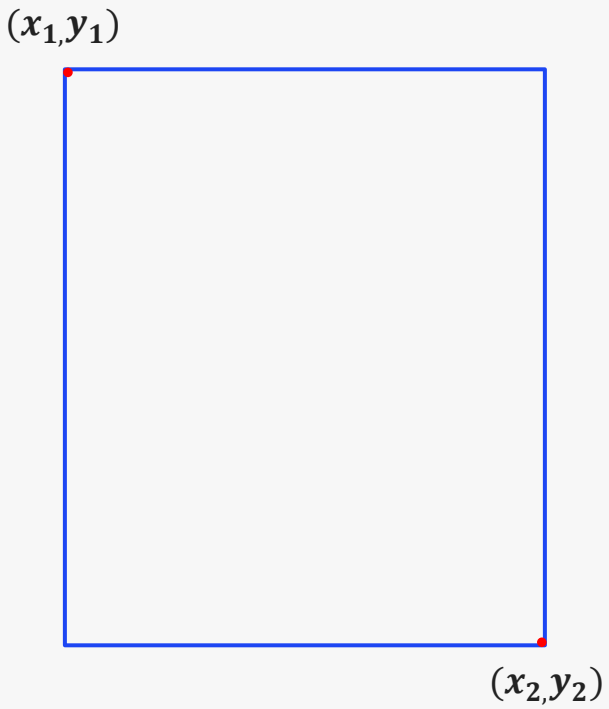
The goal of object detection is to **recognize** instances of a predefined set of object classes (e.g. {people, cars, bikes, animals}) and describe the **locations** of each detected object in the image using a ***bounding box***.

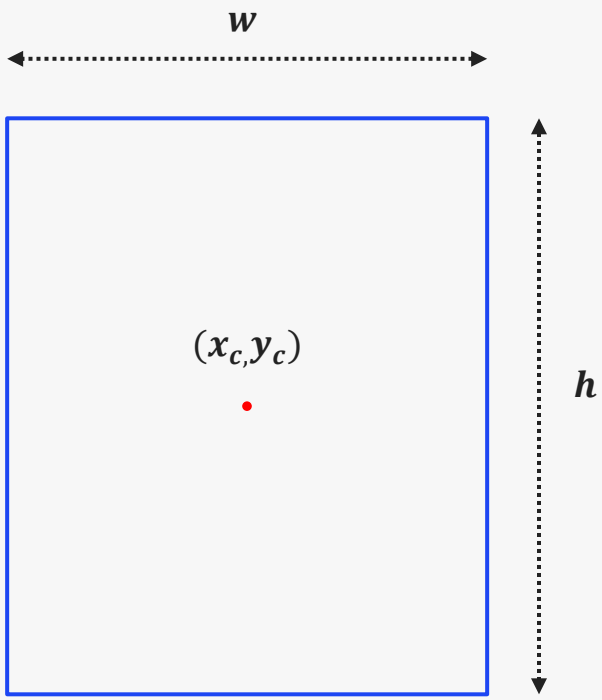
To put it simply: Object detection comes down to drawing ***bounding boxes*** around detected objects which allow us to locate them in a given scene (or how they move through it).

Object Detection: Bounding Box

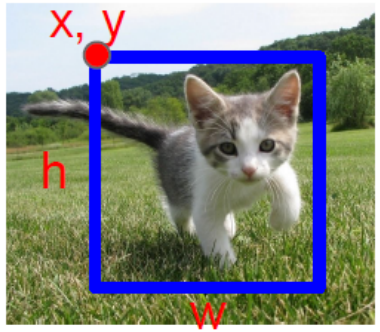
The bounding box is rectangular, which is determined by the x and y coordinates of the **upper-left** corner of the rectangle and the such coordinates of the **lower-right** corner.

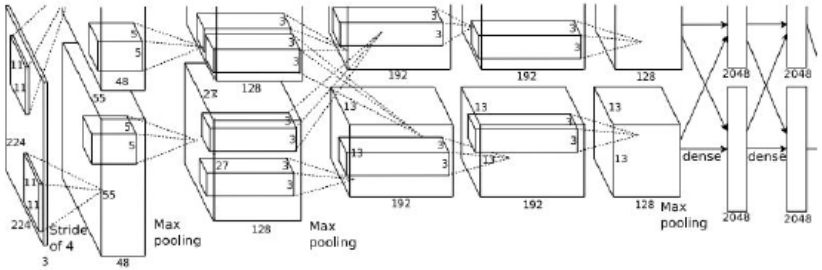
Another commonly used bounding box representation is the $(x, y) - axis$ coordinates of the bounding box **center**, and the **width** and **height** of the box.





Object Detection: Single Object:
Classification + Localization





Fully Connected: 4096 to 1000

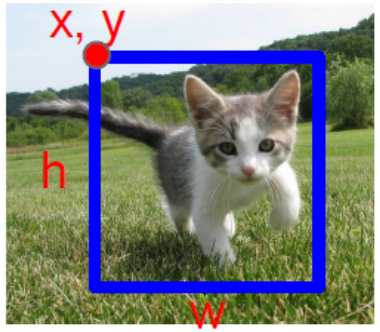
Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Vector: 4096

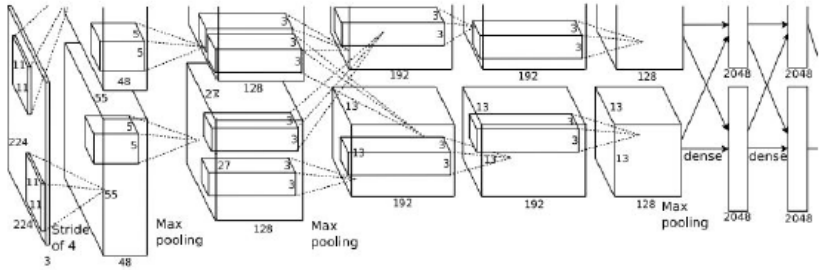
Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

Object Detection: Single Object:
Classification + Localization



This image is CC0 public domain



Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

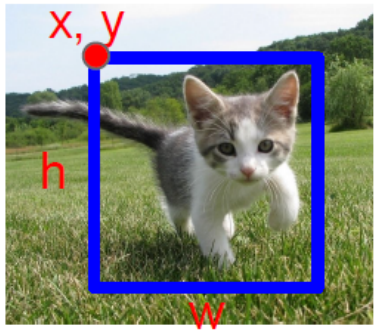
Box Coordinates (x, y, w, h)

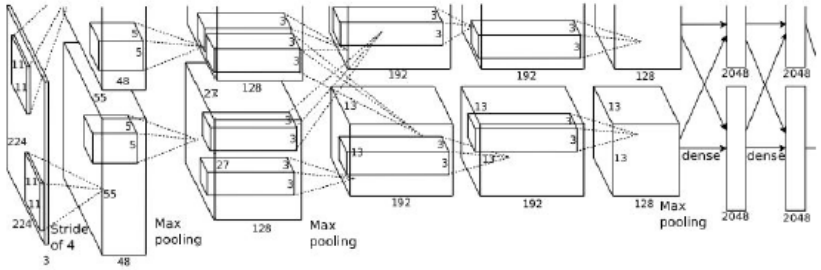
L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

Object Detection: Single Object:
Classification + Localization





Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

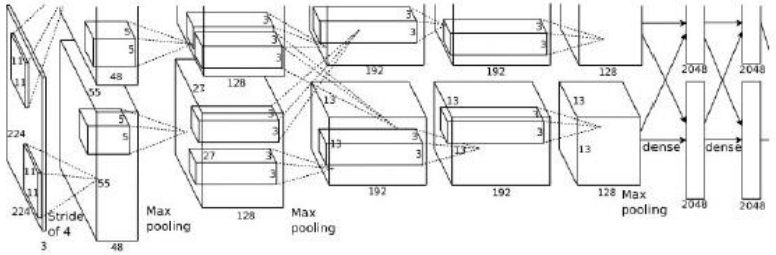
Correct box: (x', y', w', h')

Multitask Loss

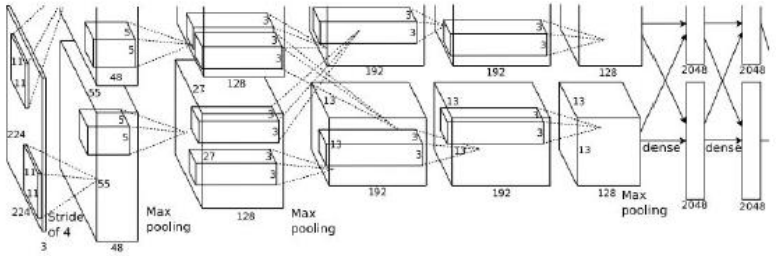
+ → **Loss**

Treat localization as a regression problem!

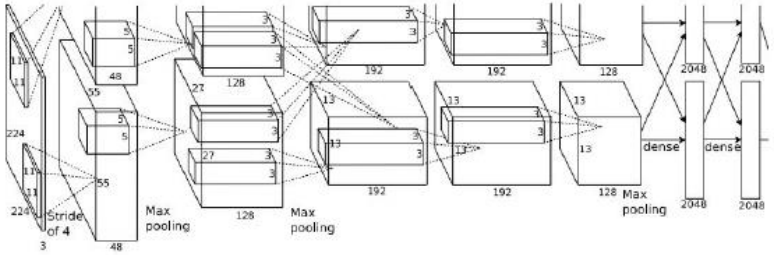
Object Detection: Multiple Objects



CAT: (x, y, w, h)



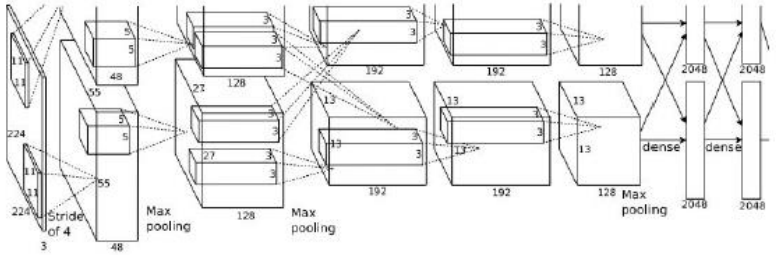
DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

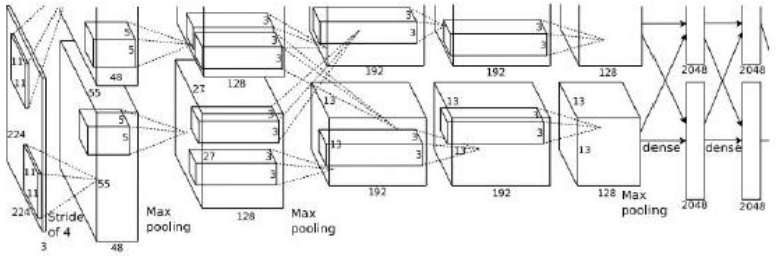
Object Detection: Multiple Objects

Each image needs a different number of outputs!



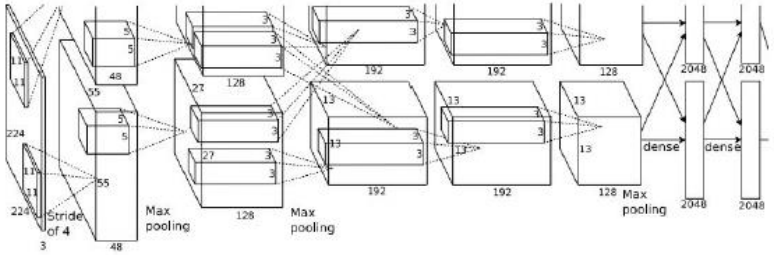
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

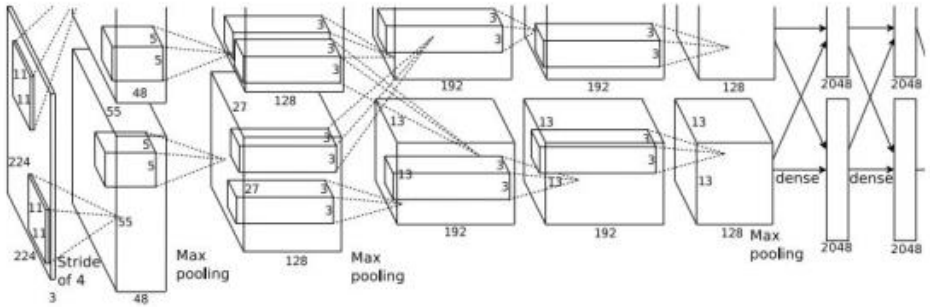


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Object Detection: Multiple Objects

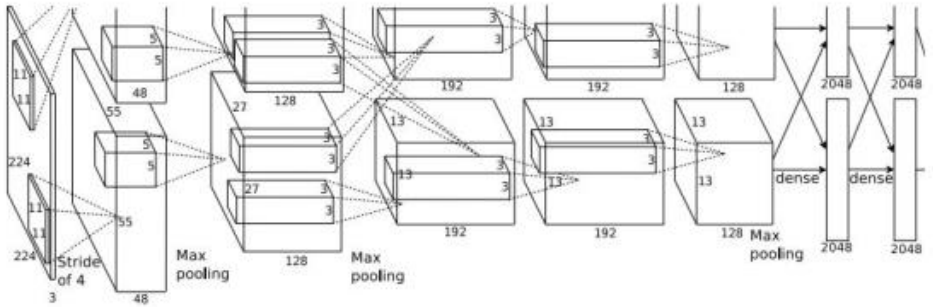
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

Object Detection: Multiple Objects

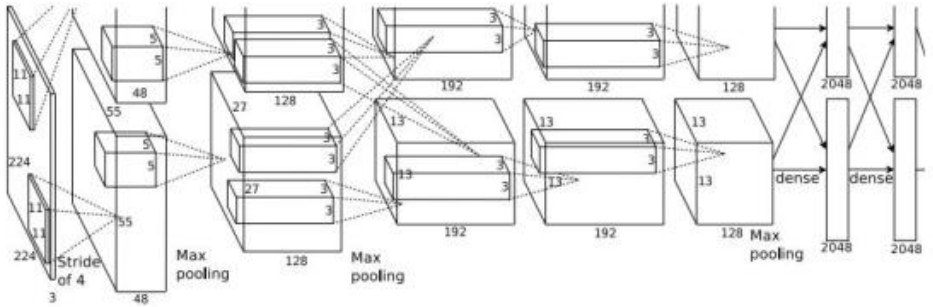
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

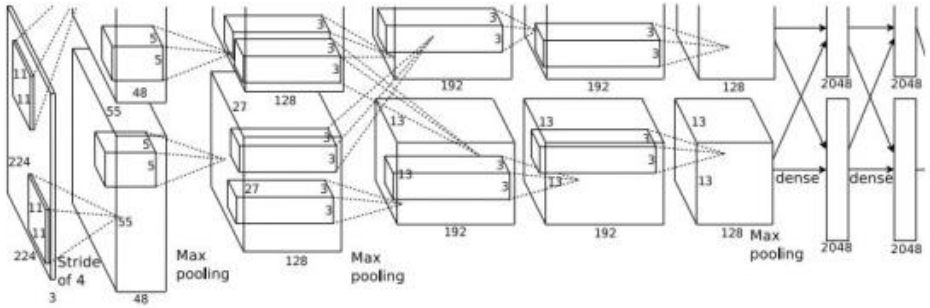
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

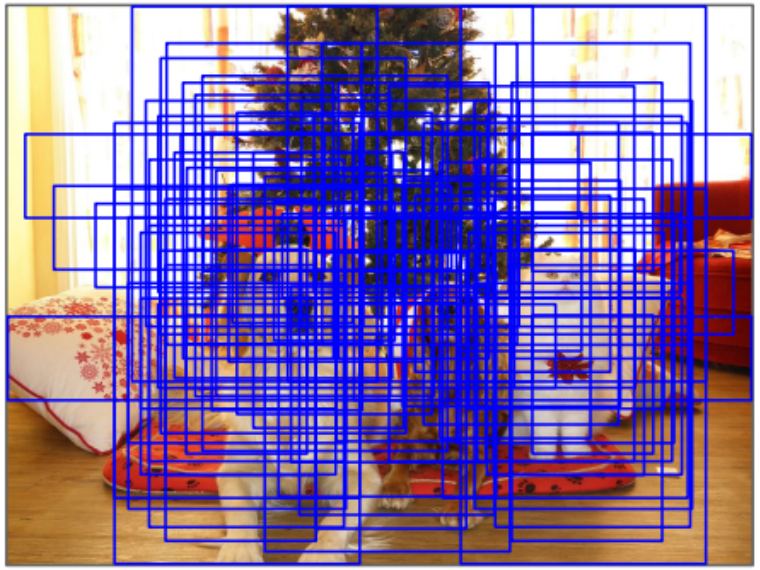
Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

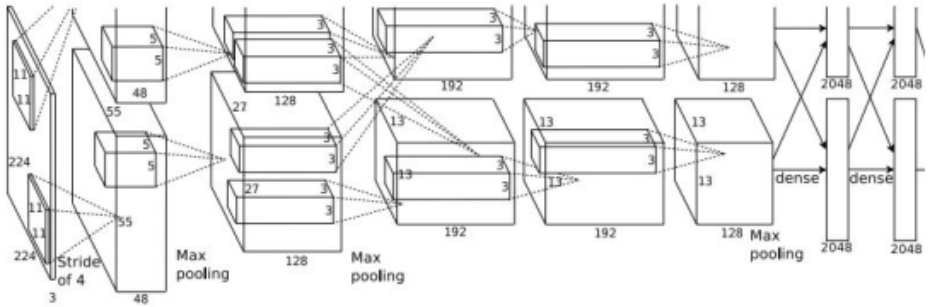


Dog? NO
Cat? YES
Background? NO

Object Detection: Multiple Objects



Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

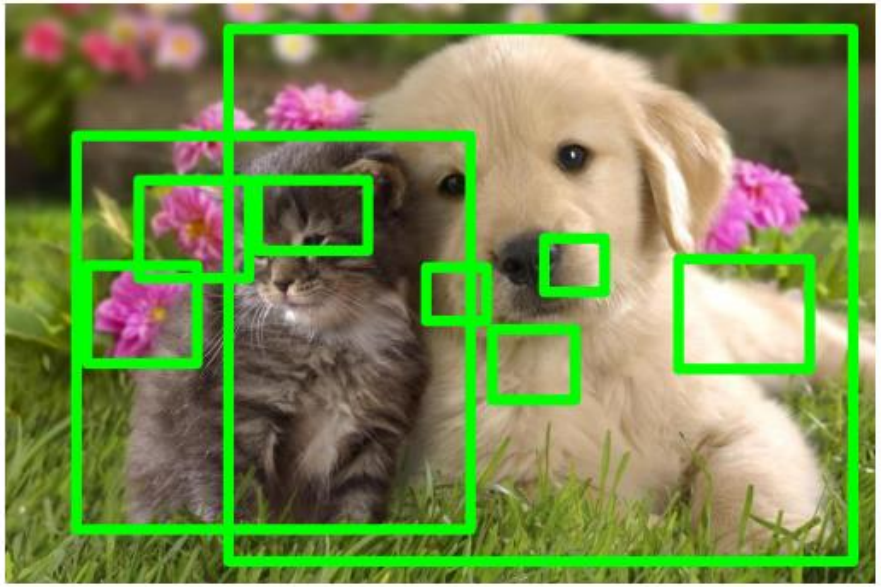


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



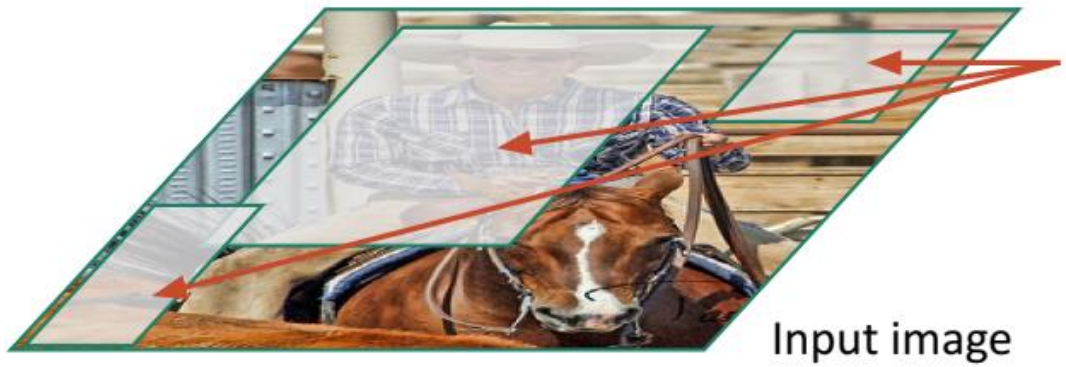
Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

Object Detection: R-CNN

Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

Object Detection: R-CNN

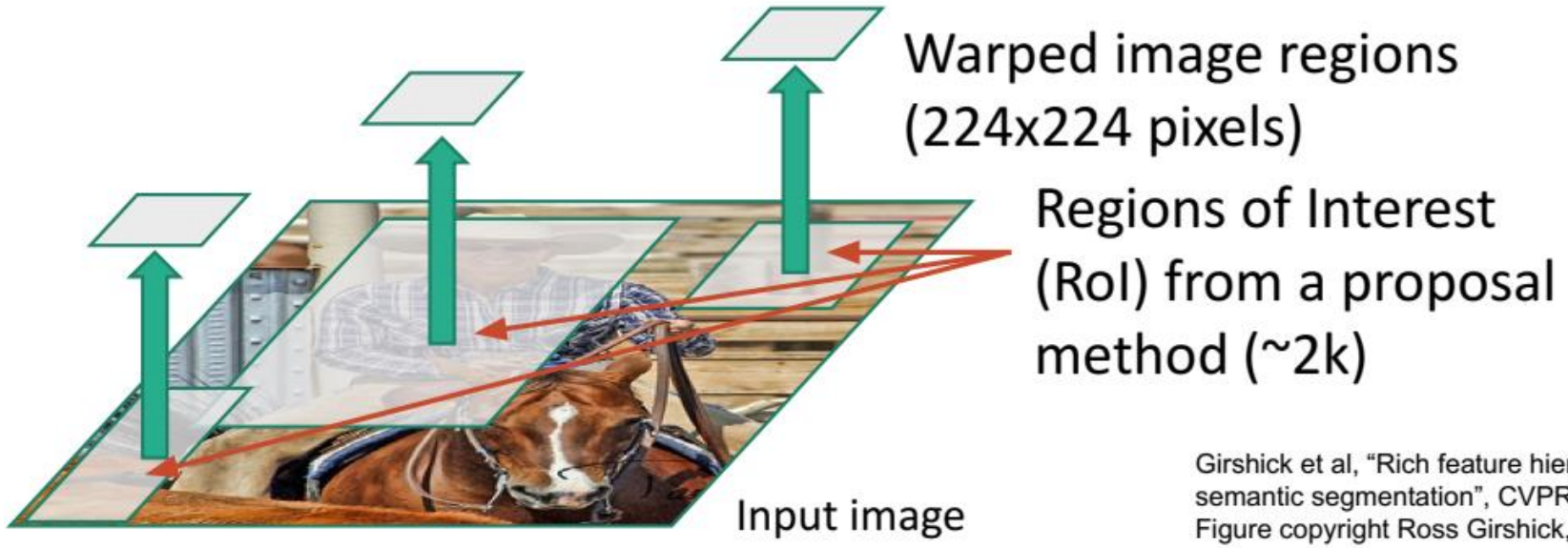


Input image

Regions of Interest (RoI) from a proposal method (~2k)

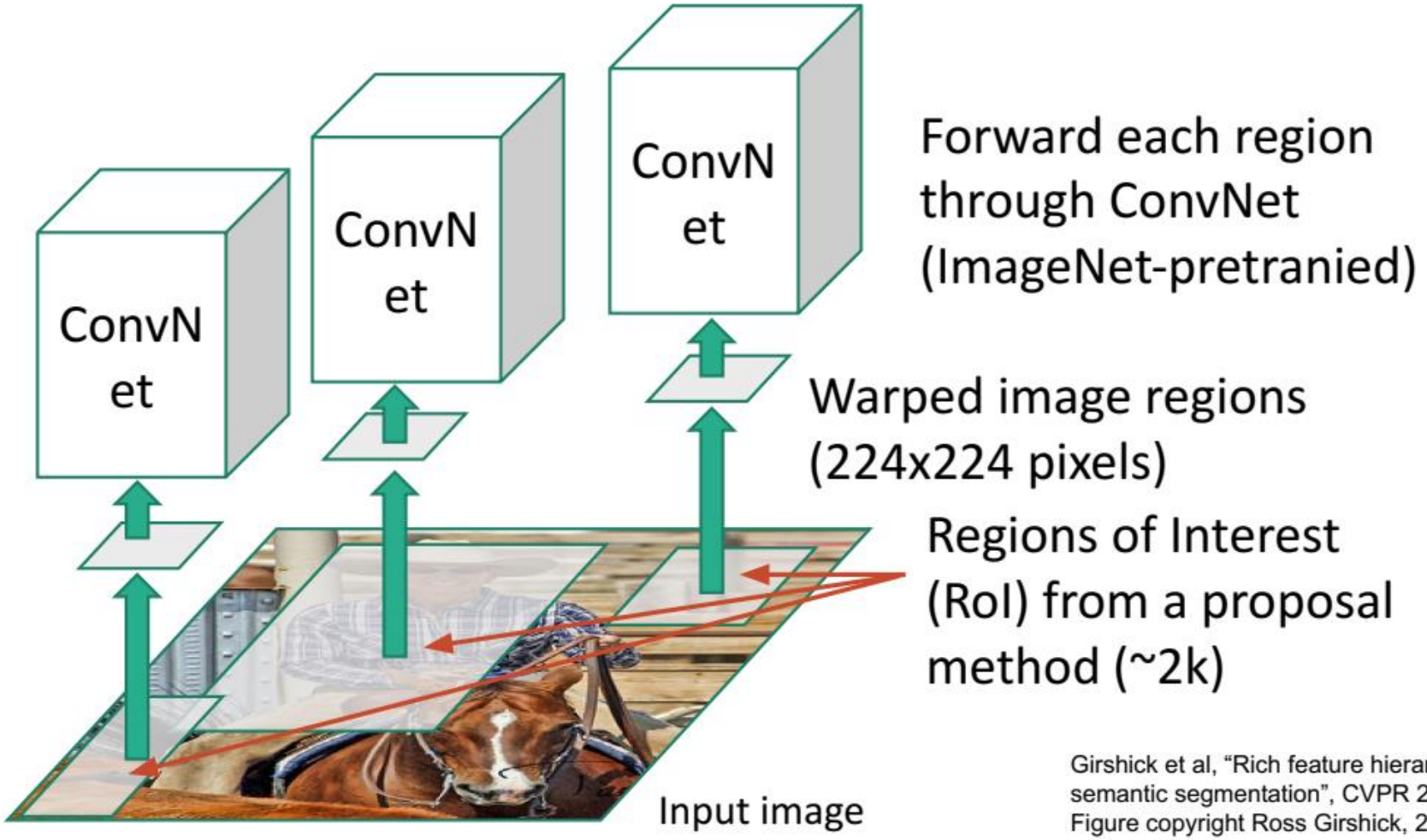
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Object Detection: R-CNN



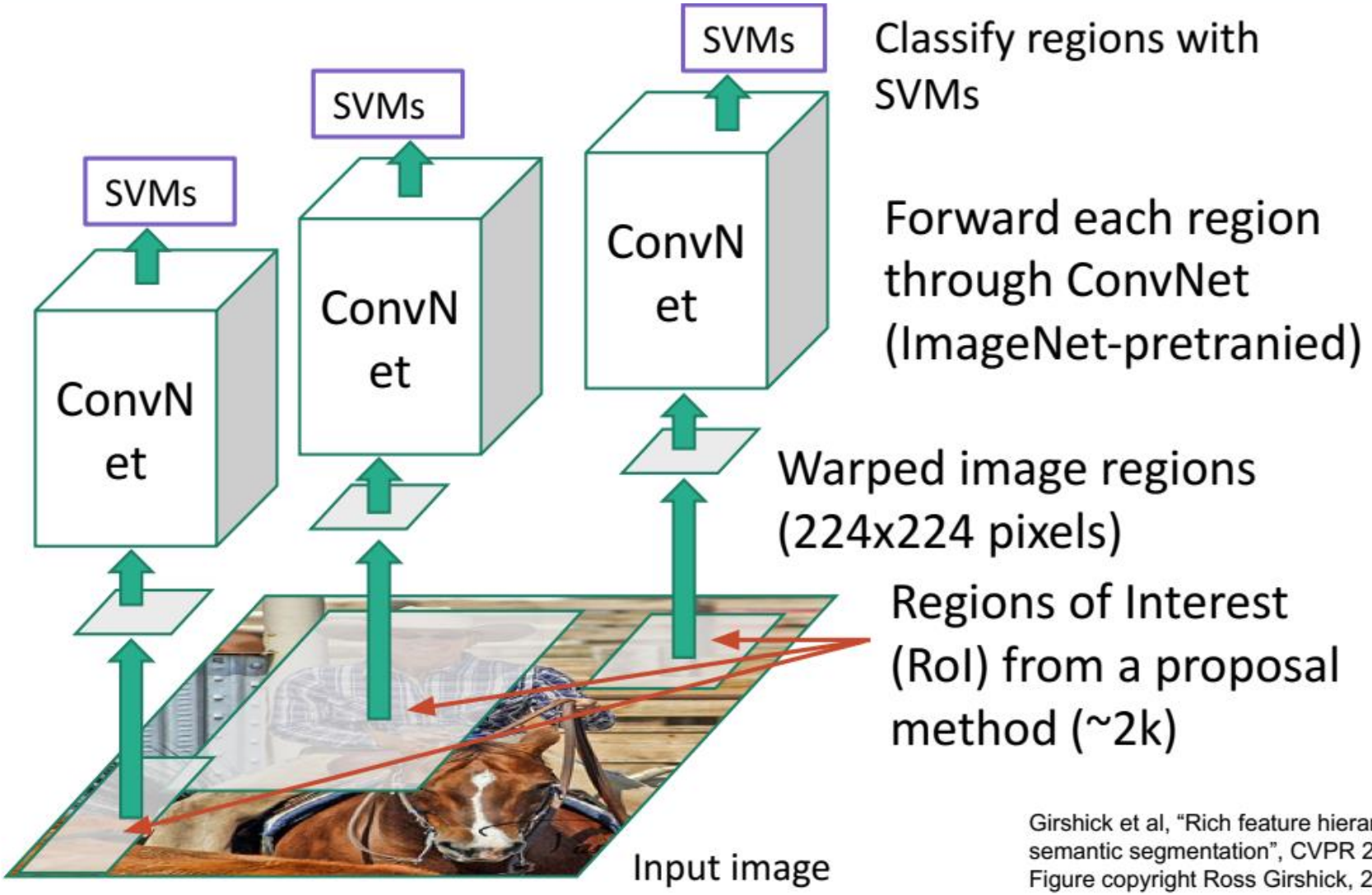
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Object Detection: R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

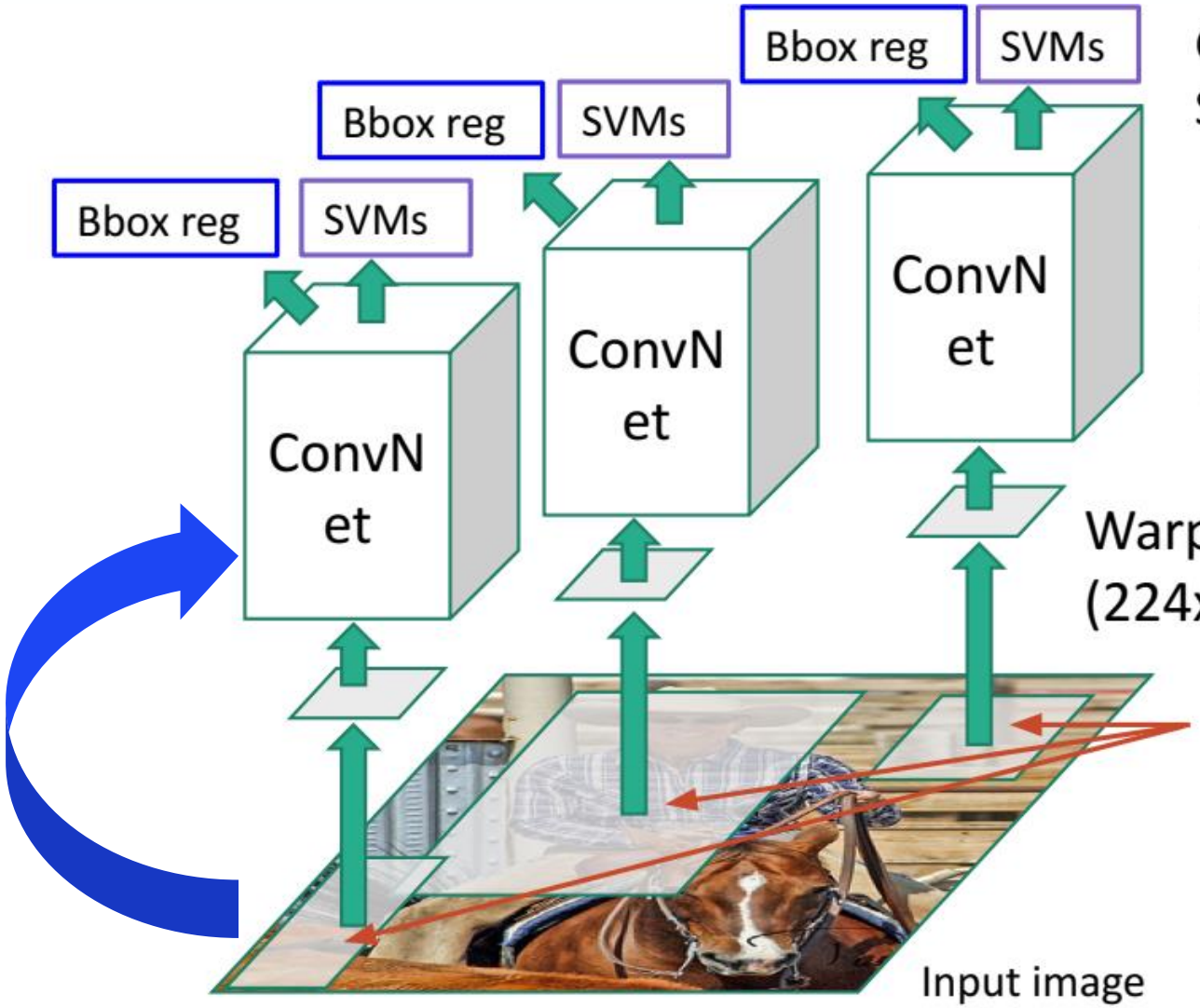
Object Detection: R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Object Detection: R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet (ImageNet-pretranied)

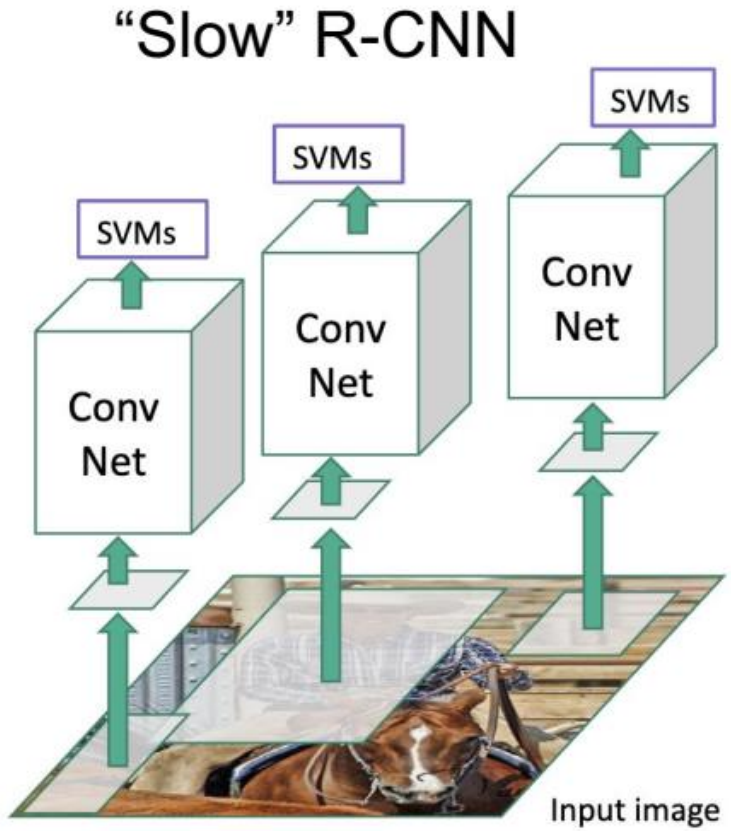
Warped image regions (224x224 pixels)

Regio
(RoI) f
meth

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

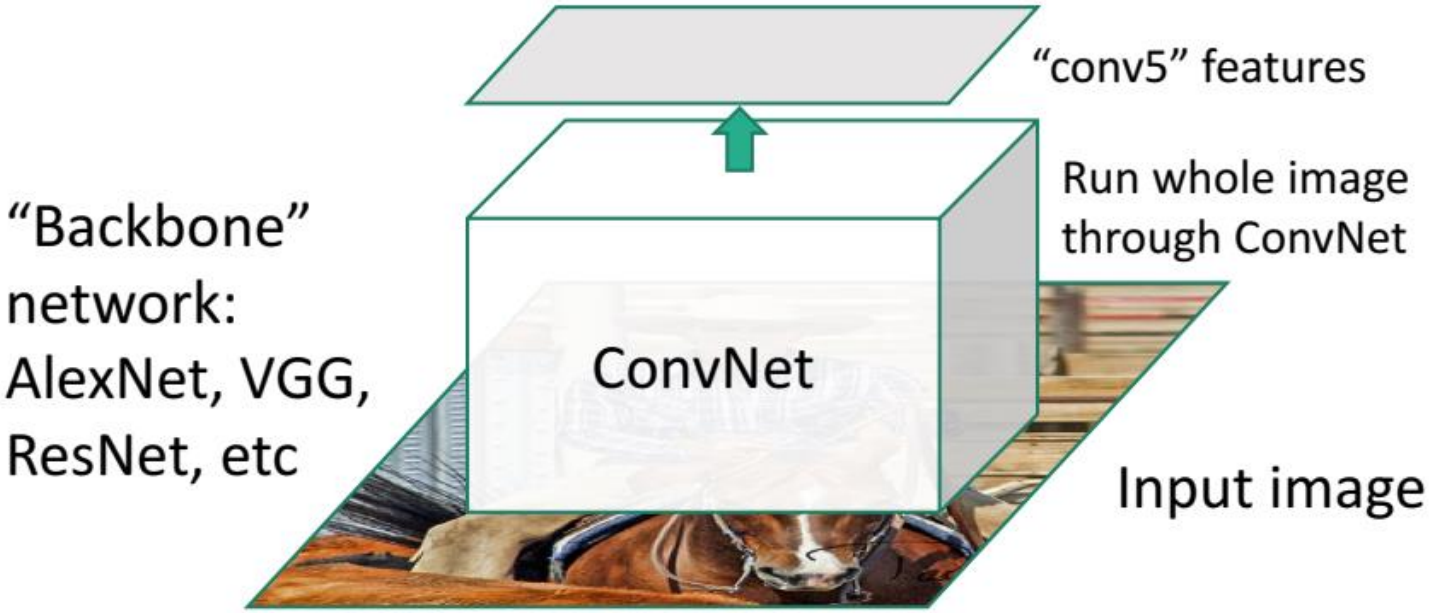
Problem: Very slow!
Need to do ~2k independent forward passes for each image!

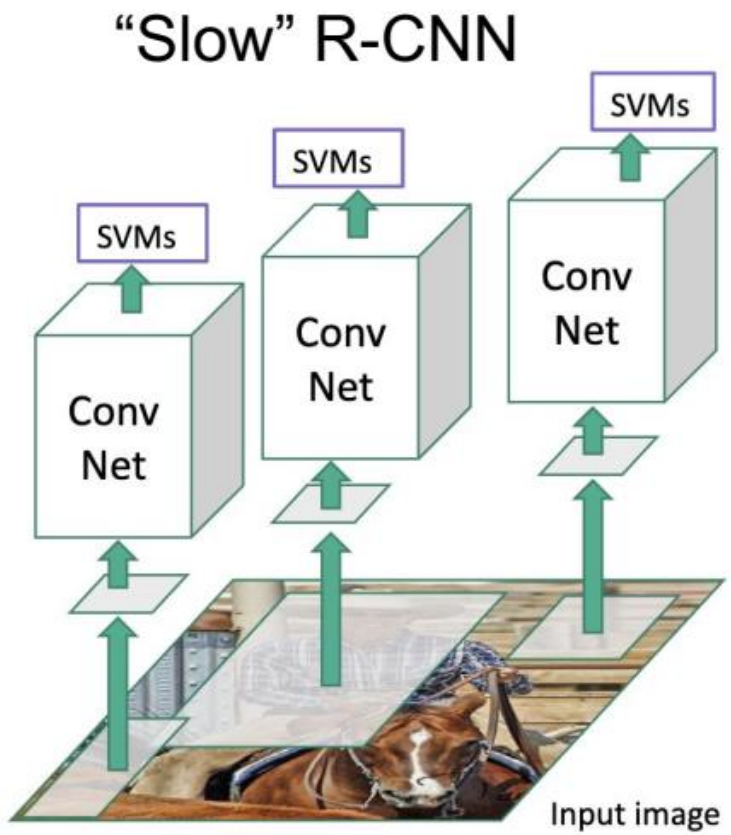
Object Detection: Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

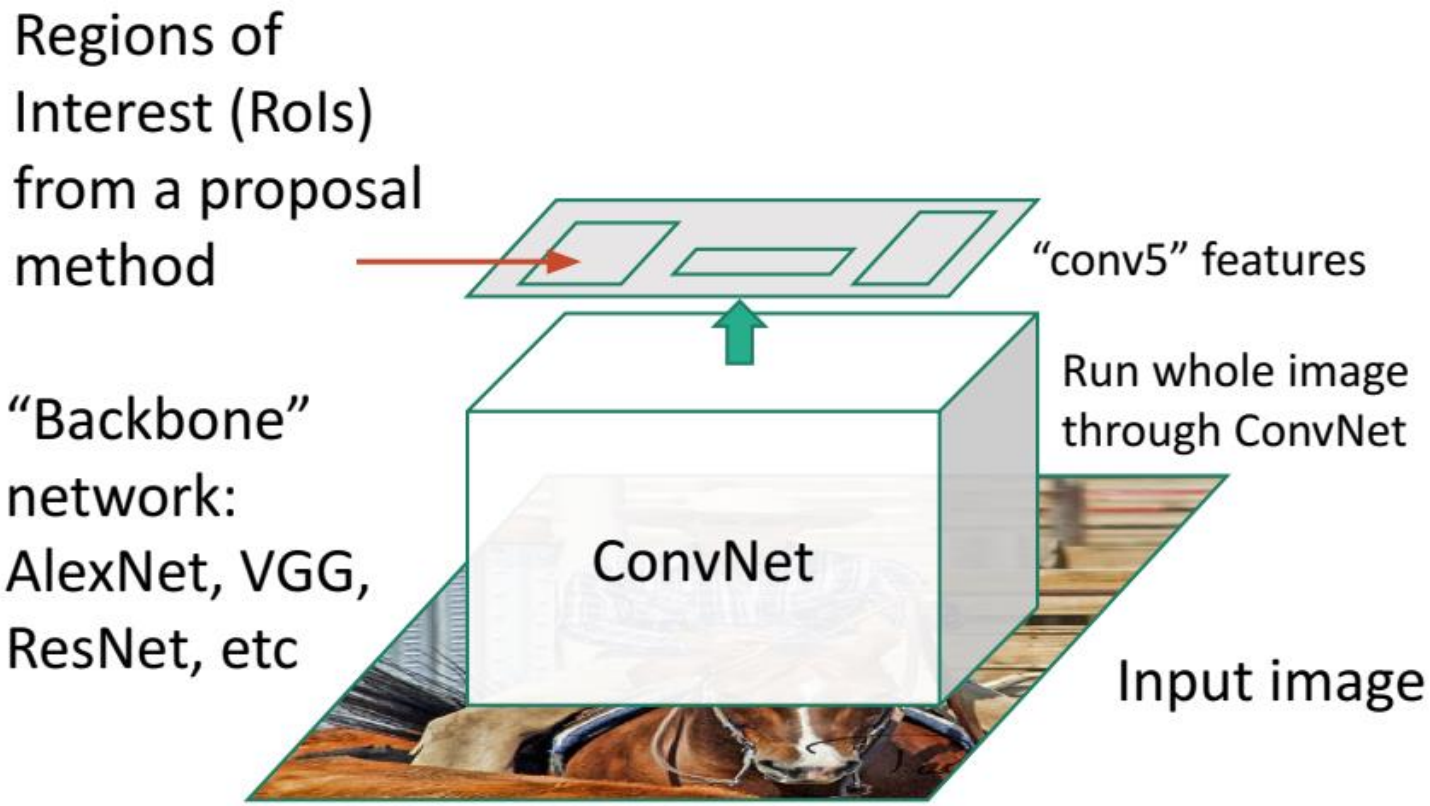
Object Detection: Fast R-CNN

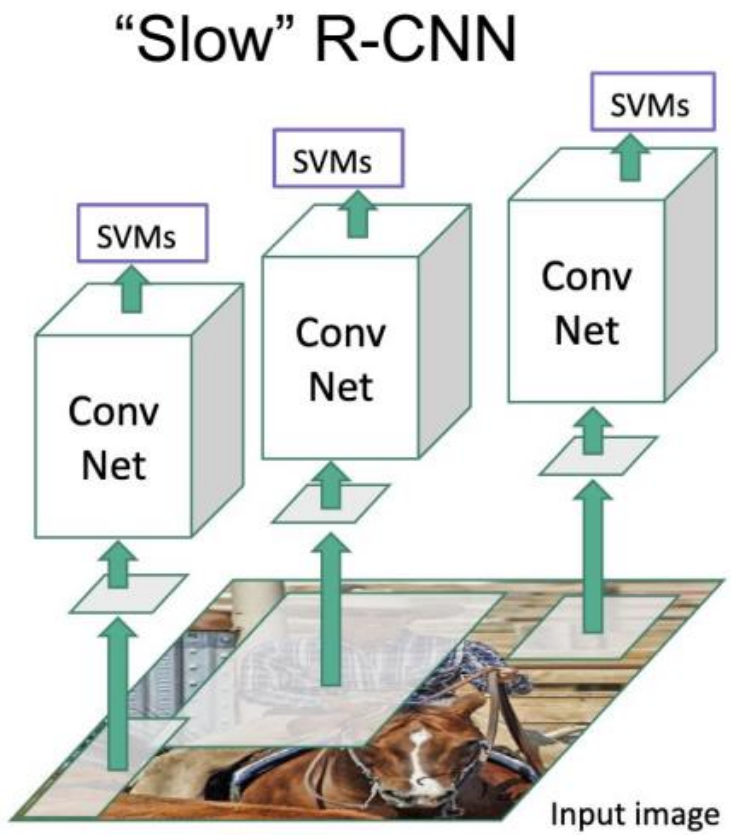




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

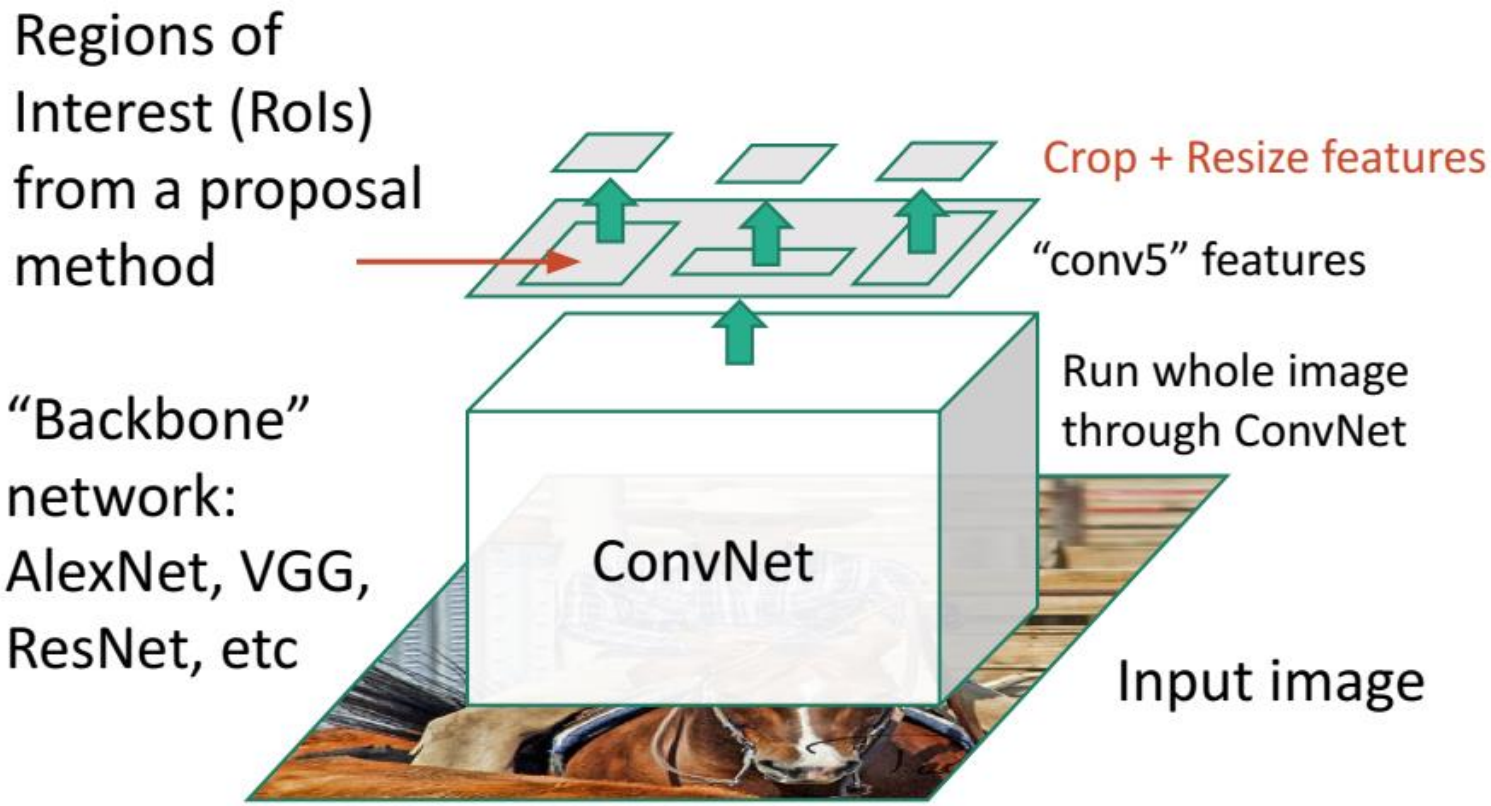
Object Detection: Fast R-CNN

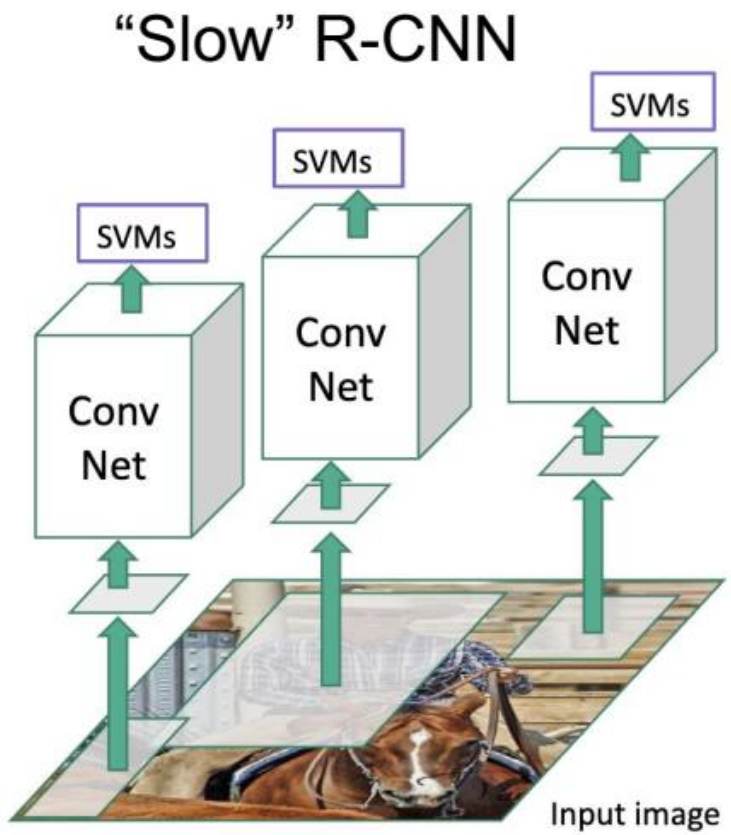




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

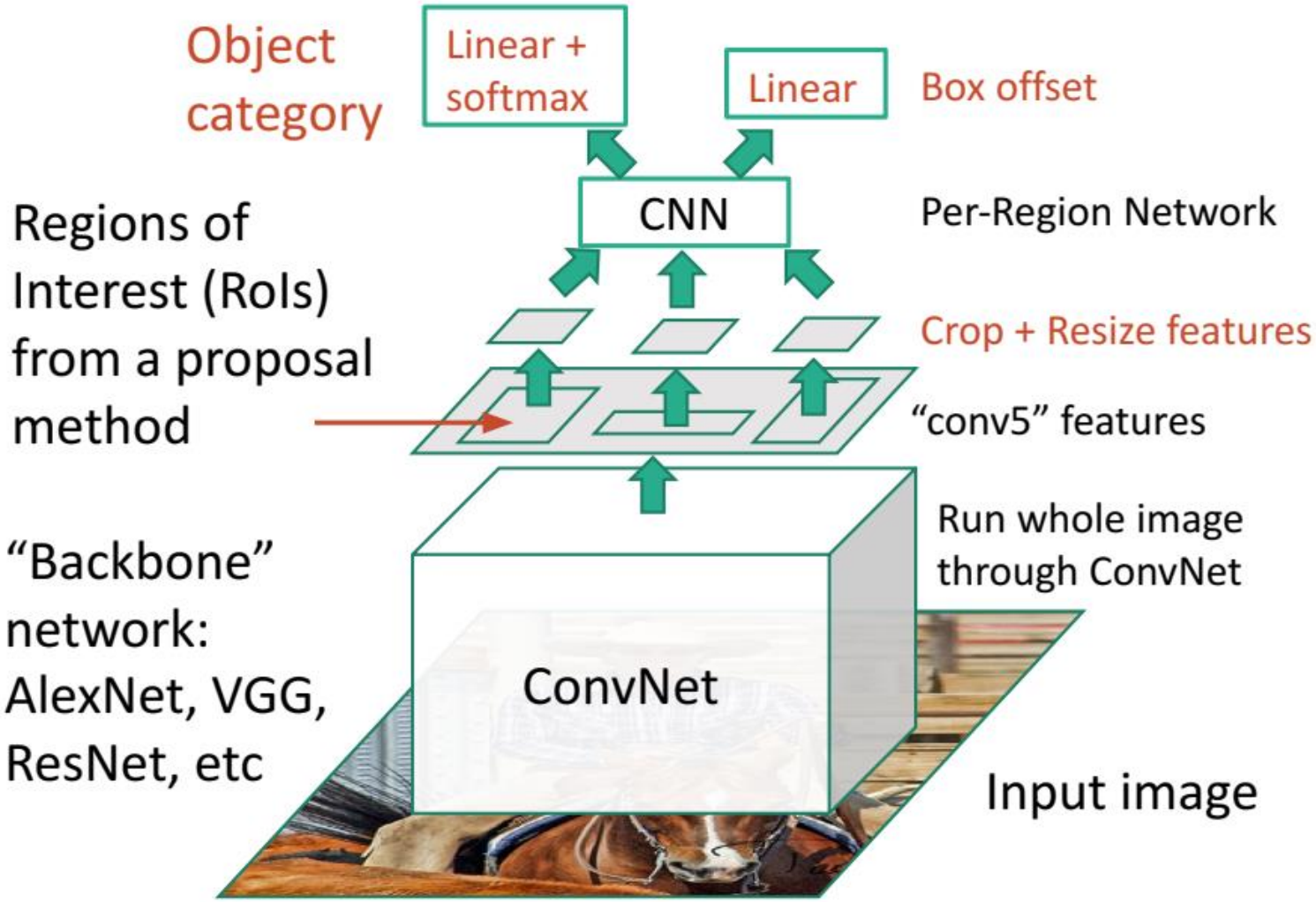
Object Detection: Fast R-CNN

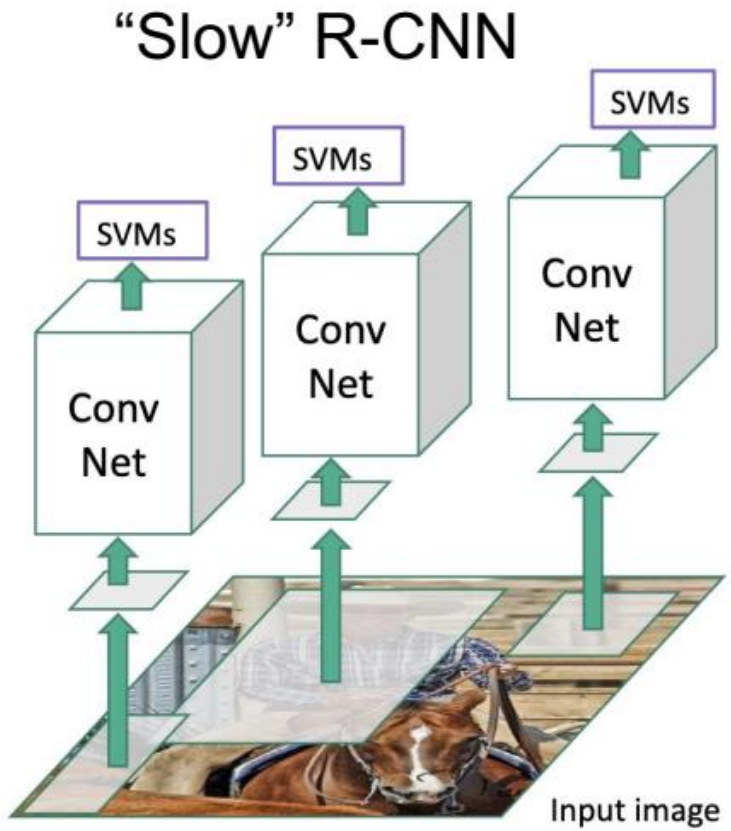




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

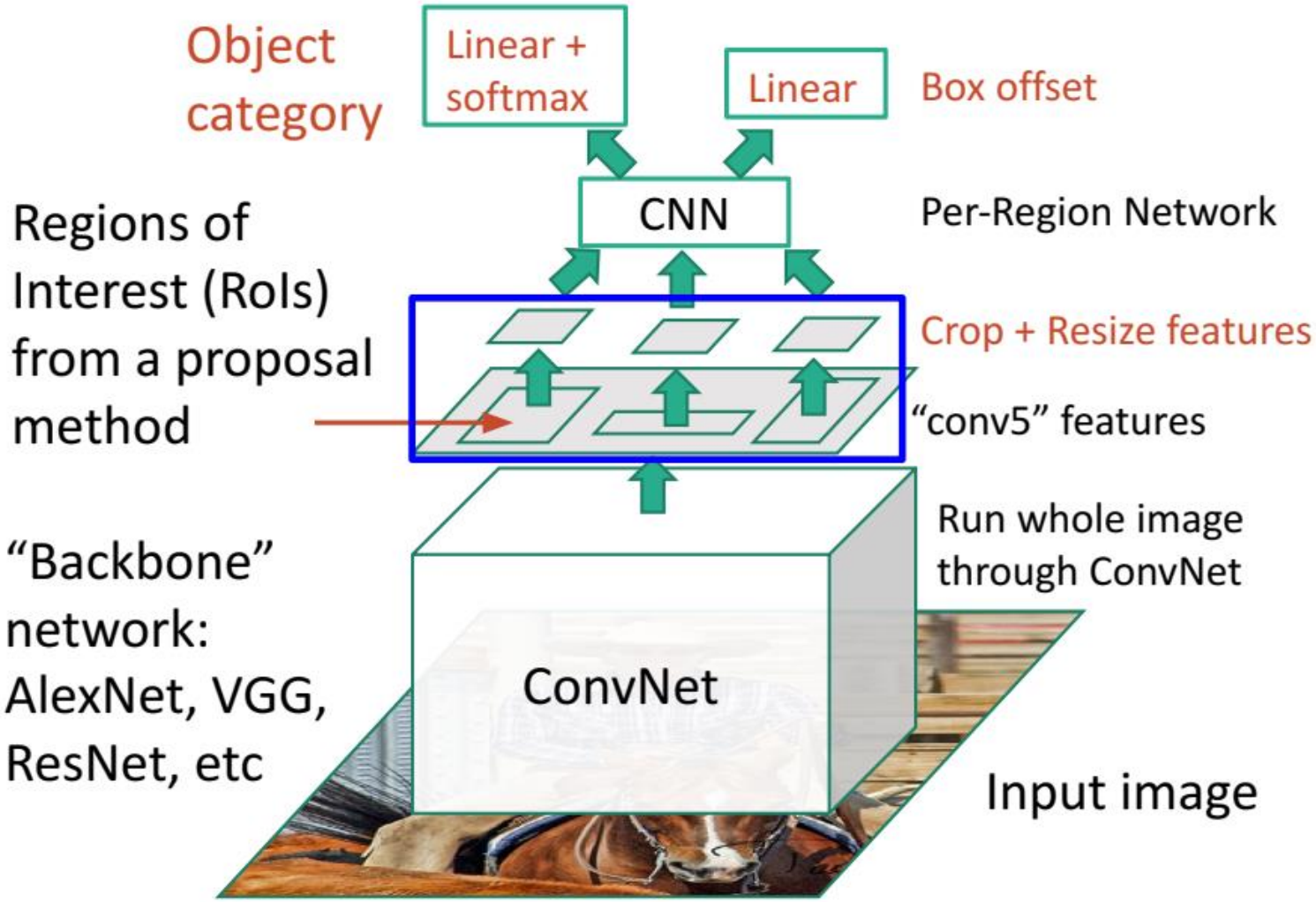
Object Detection: Fast R-CNN

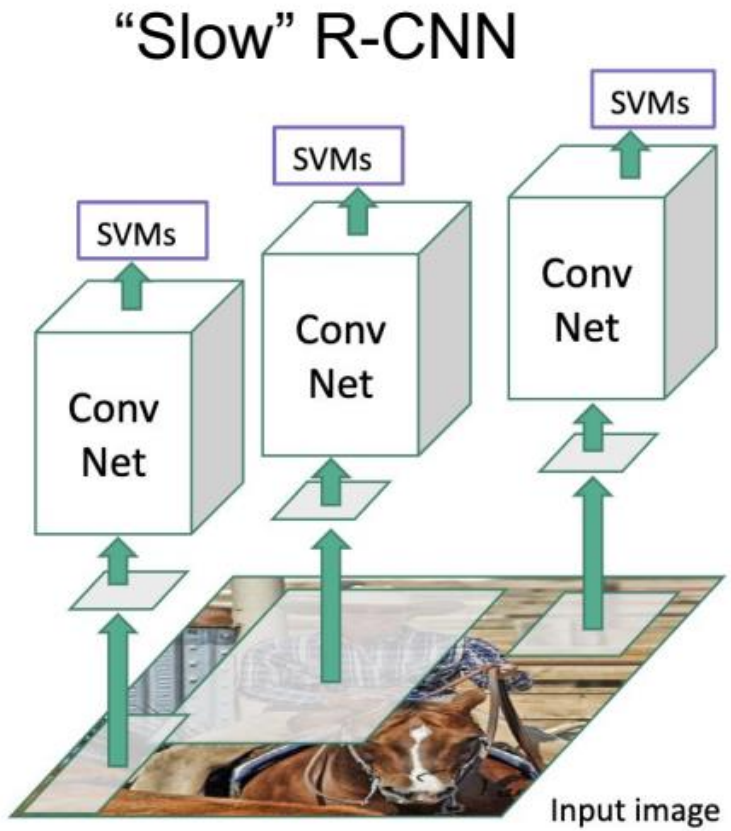




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

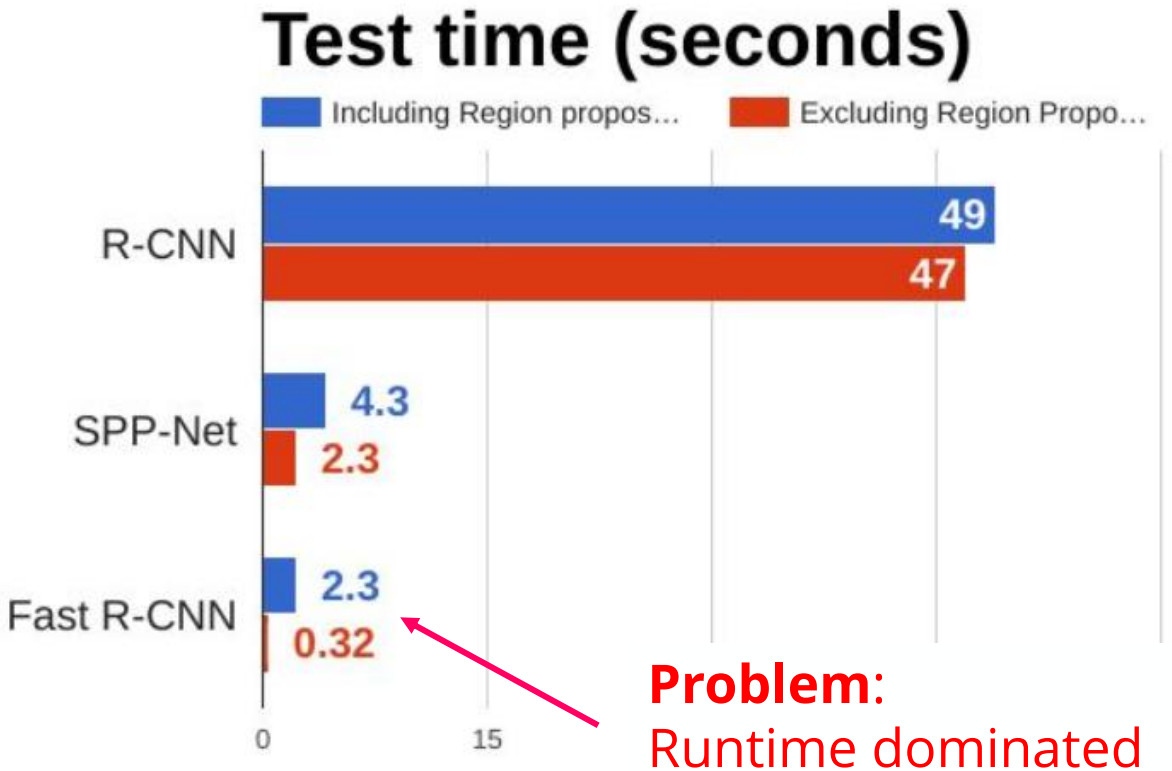
Object Detection: Fast R-CNN





Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs. Fast R-CNN



Problem:
Runtime dominated by region proposals!

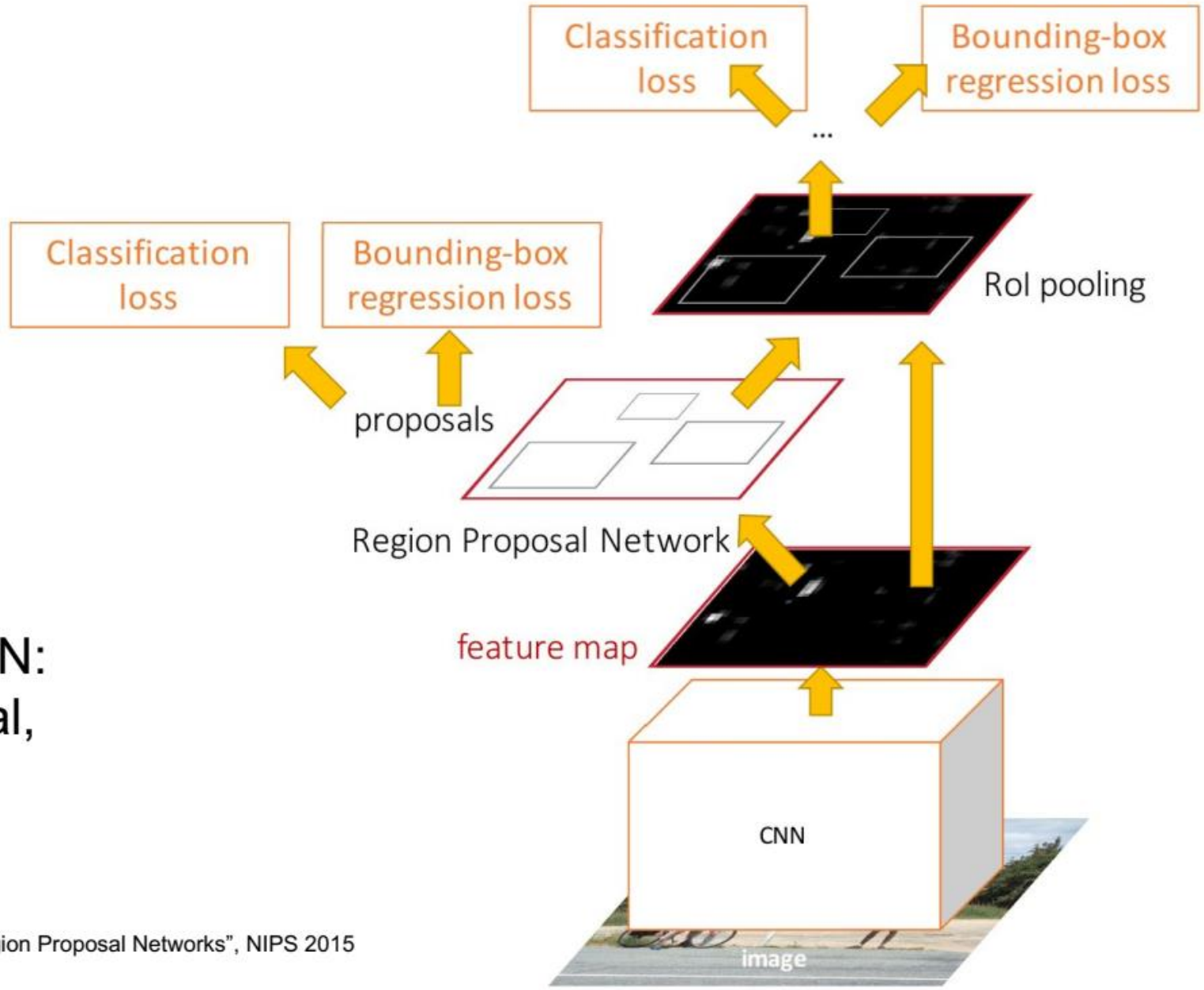
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

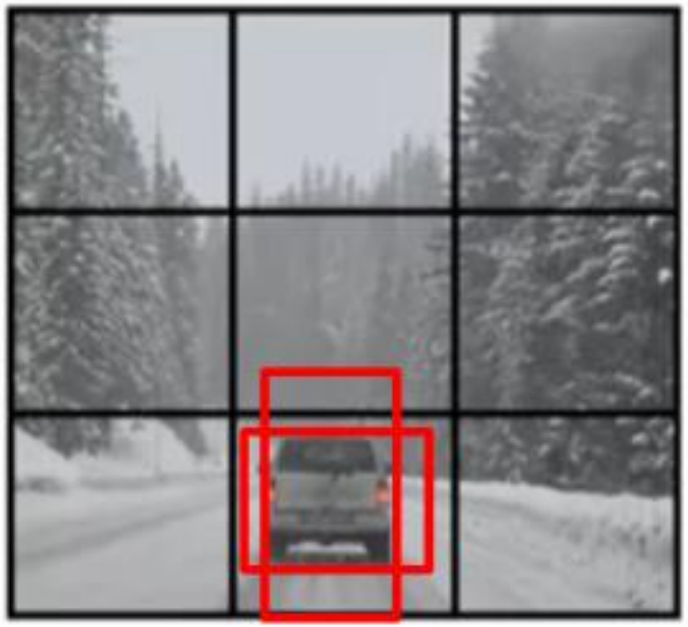
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Anchor Boxes

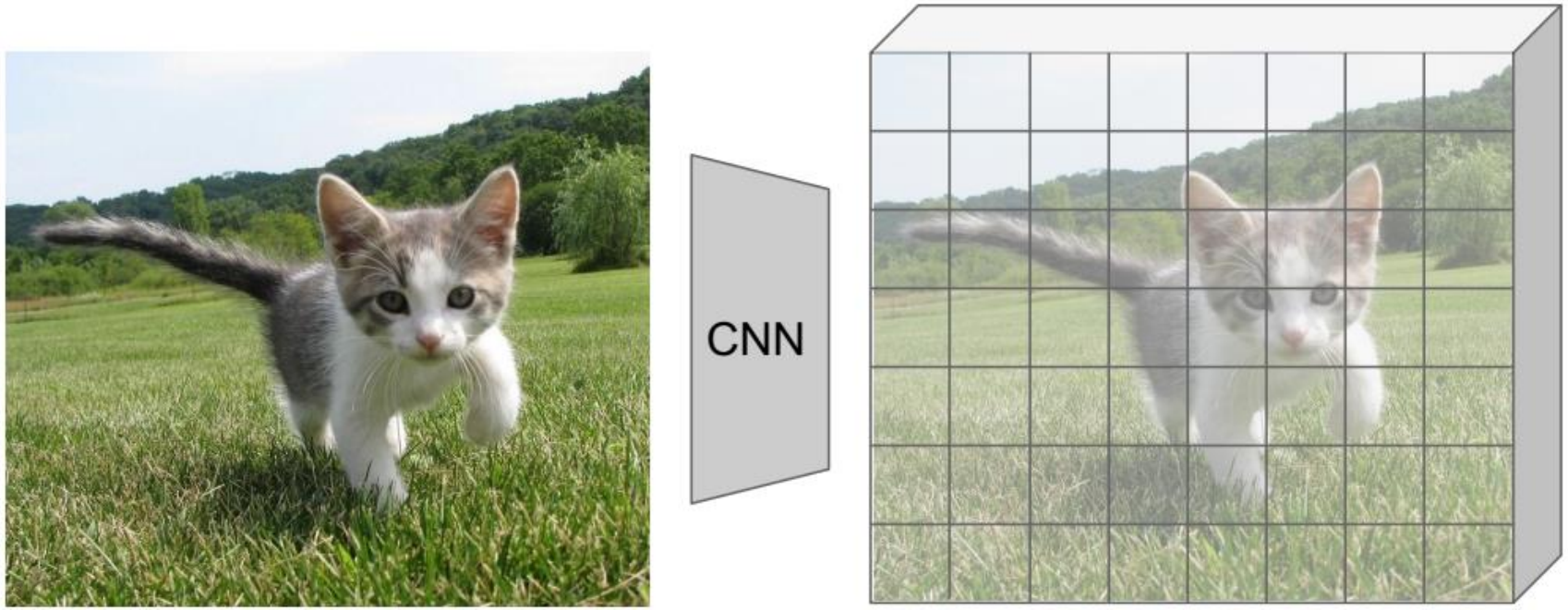
Anchor Boxes are multiple bounding with varying scales and aspect ratios centered on each pixel.



Anchor box 1

Anchor box 2

Region Proposal Network (RPN)



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512 x 20 x 15)

Region Proposal Network (RPN)

Input Image
(e.g. 3 x 640 x 480)

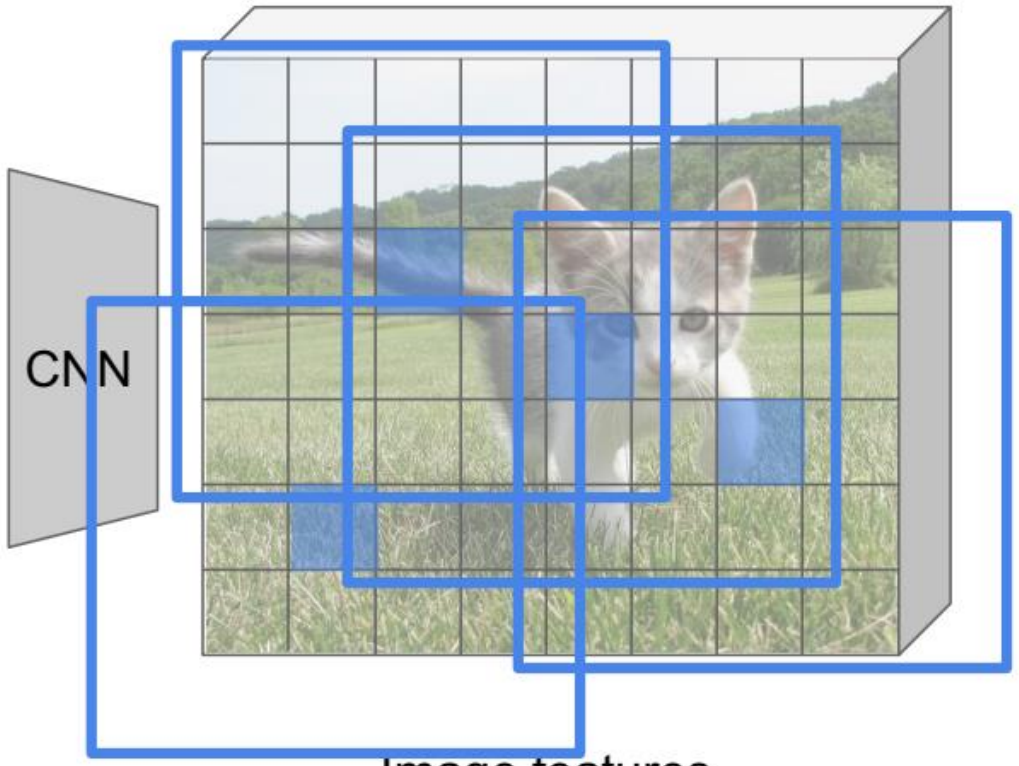


Image features
(e.g. 512 x 20 x 15)

Imagine an anchor box of fixed size at each point in the feature map

Region Proposal Network (RPN)

Input Image
(e.g. 3 x 640 x 480)

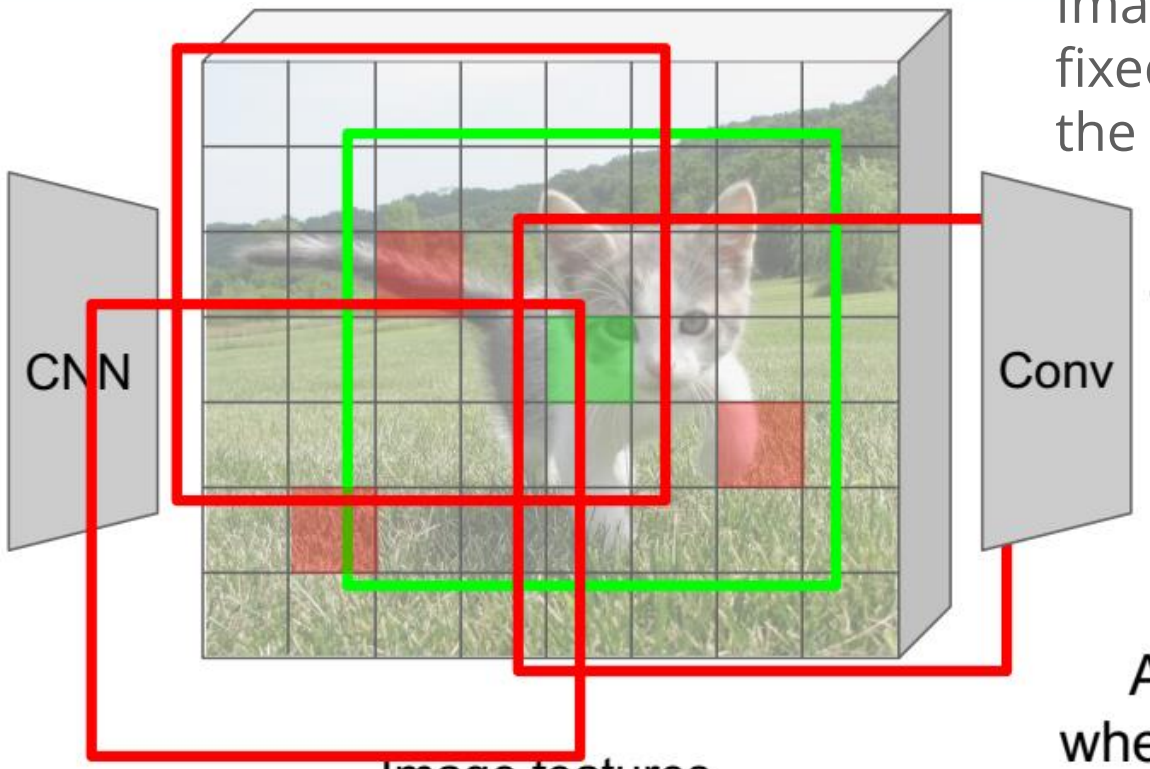


Image features
(e.g. 512 x 20 x 15)

Imagine an anchor box of fixed size at each point in the feature map

Anchor is an object?
1 x 20 x 15

At each point, predict whether the corresponding anchor contains an object (binary classification)

Region Proposal Network (RPN)

Input Image
(e.g. 3 x 640 x 480)

CNN

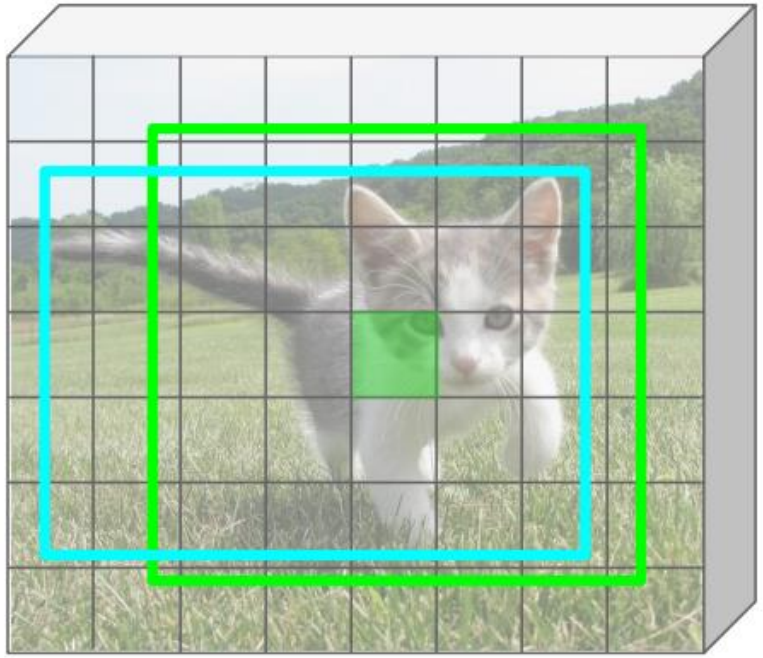


Image features
(e.g. 512 x 20 x 15)

Imagine an anchor box of fixed size at each point in the feature map

Conv

Anchor is an object?
1 x 20 x 15

Box corrections
4 x 20 x 15

For positive boxes, also predict a corrections from the anchor to the ground-truth box (regress 4 numbers per pixel)

Region Proposal Network (RPN)

Input Image
(e.g. 3 x 640 x 480)

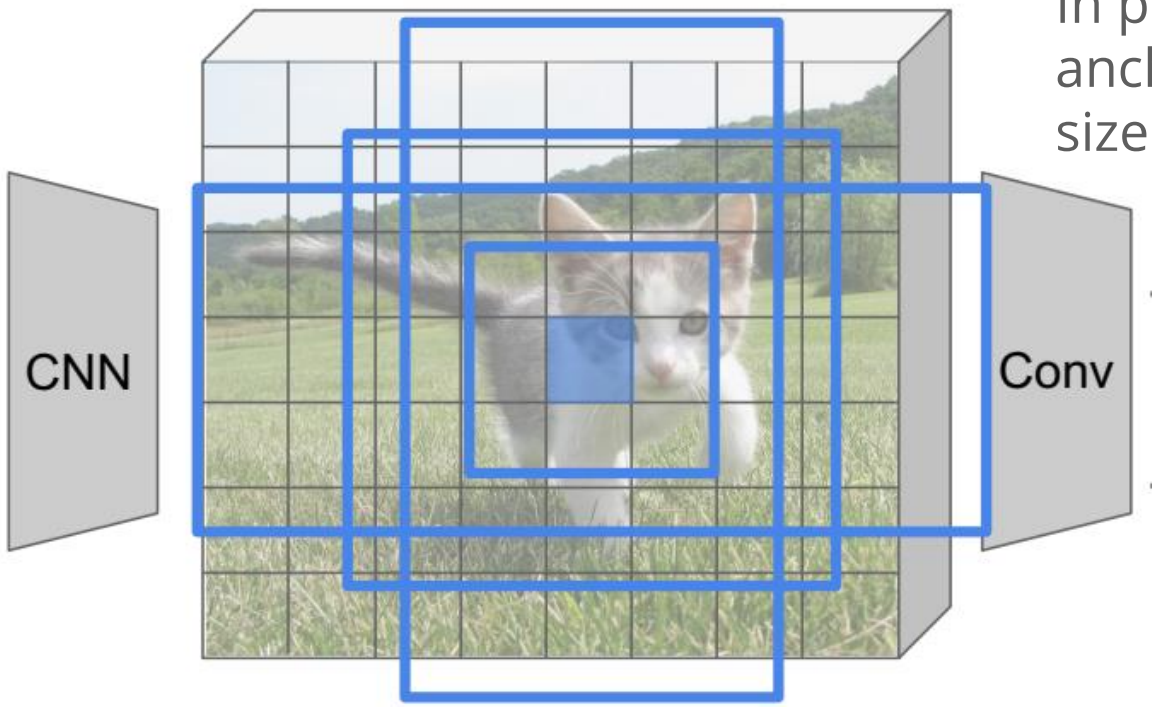


Image features
(e.g. 512 x 20 x 15)

In practice use K different anchor boxes of different size / scale at each point

Anchor is an object?
K x 20 x 15

Box transforms
4K x 20 x 15

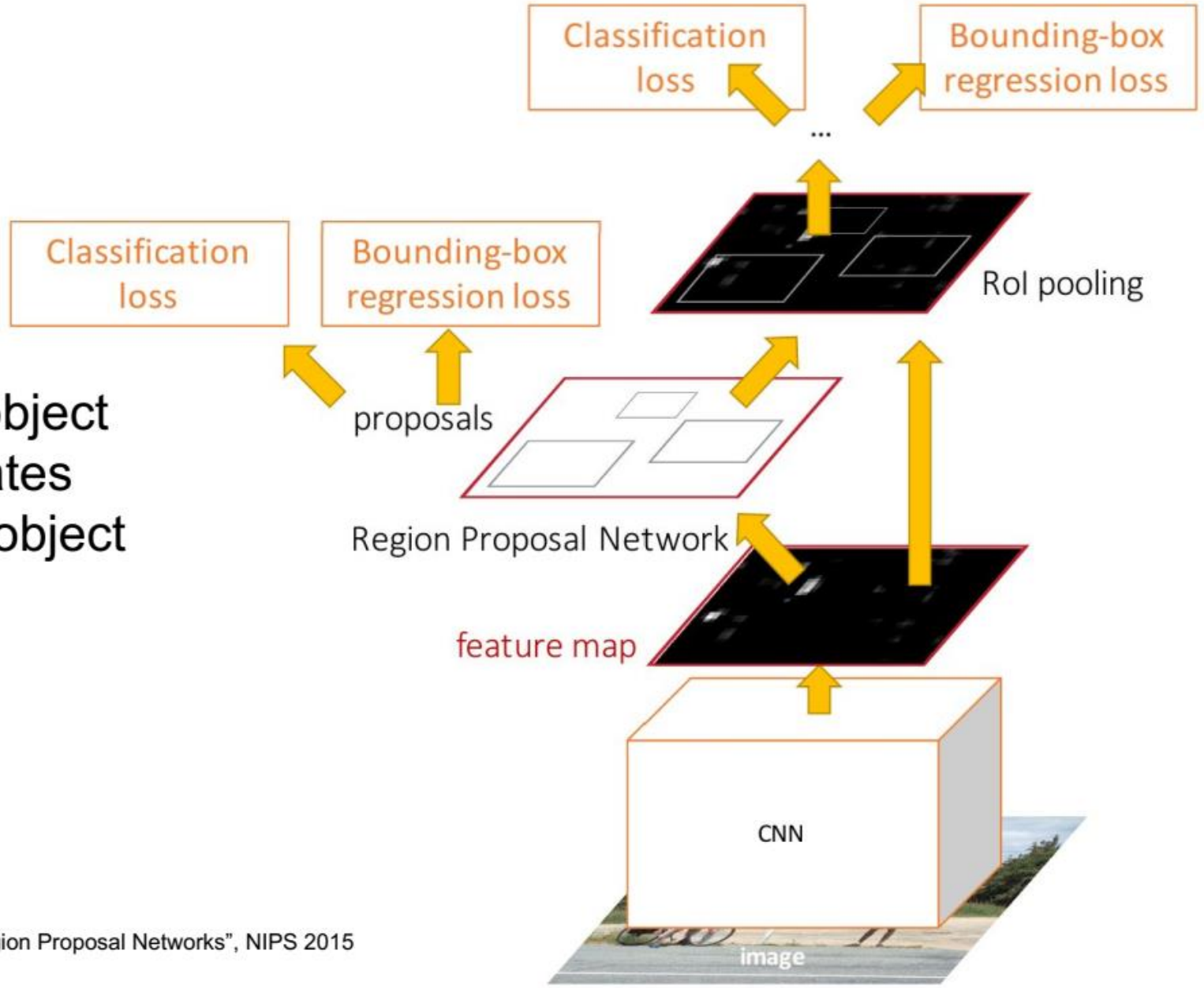
Sort the K*20*15 boxes by their “objectness” score, take top ~300 as our proposals

Faster R-CNN

Make CNN do proposals!

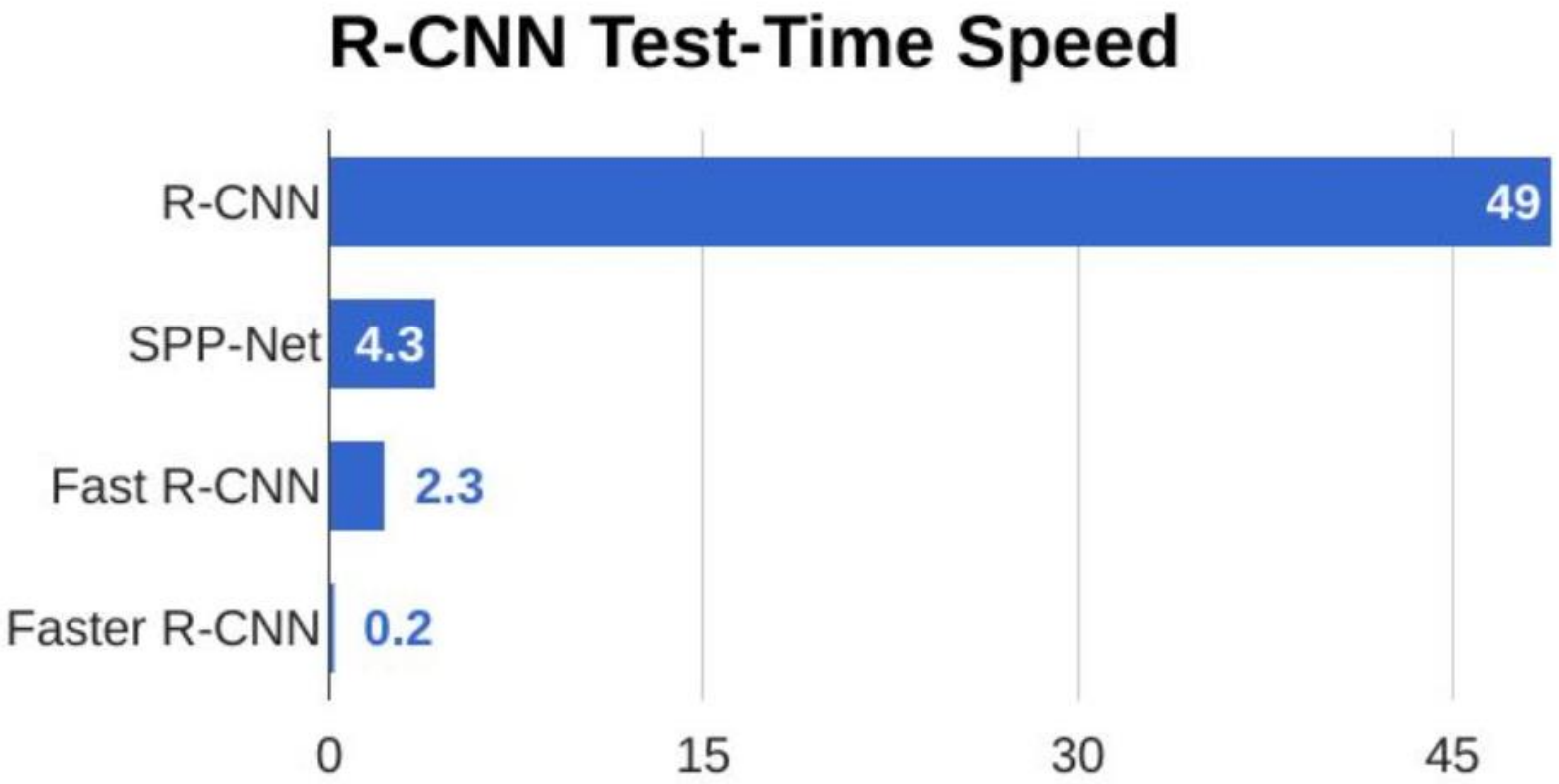
Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN


R-CNN Test-Time Speed
R-CNN
SPP-Net
Fast R-CNN
Faster R-CNN
49
4.3
2.3
0.2
0
15
30
45

Faster R-CNN

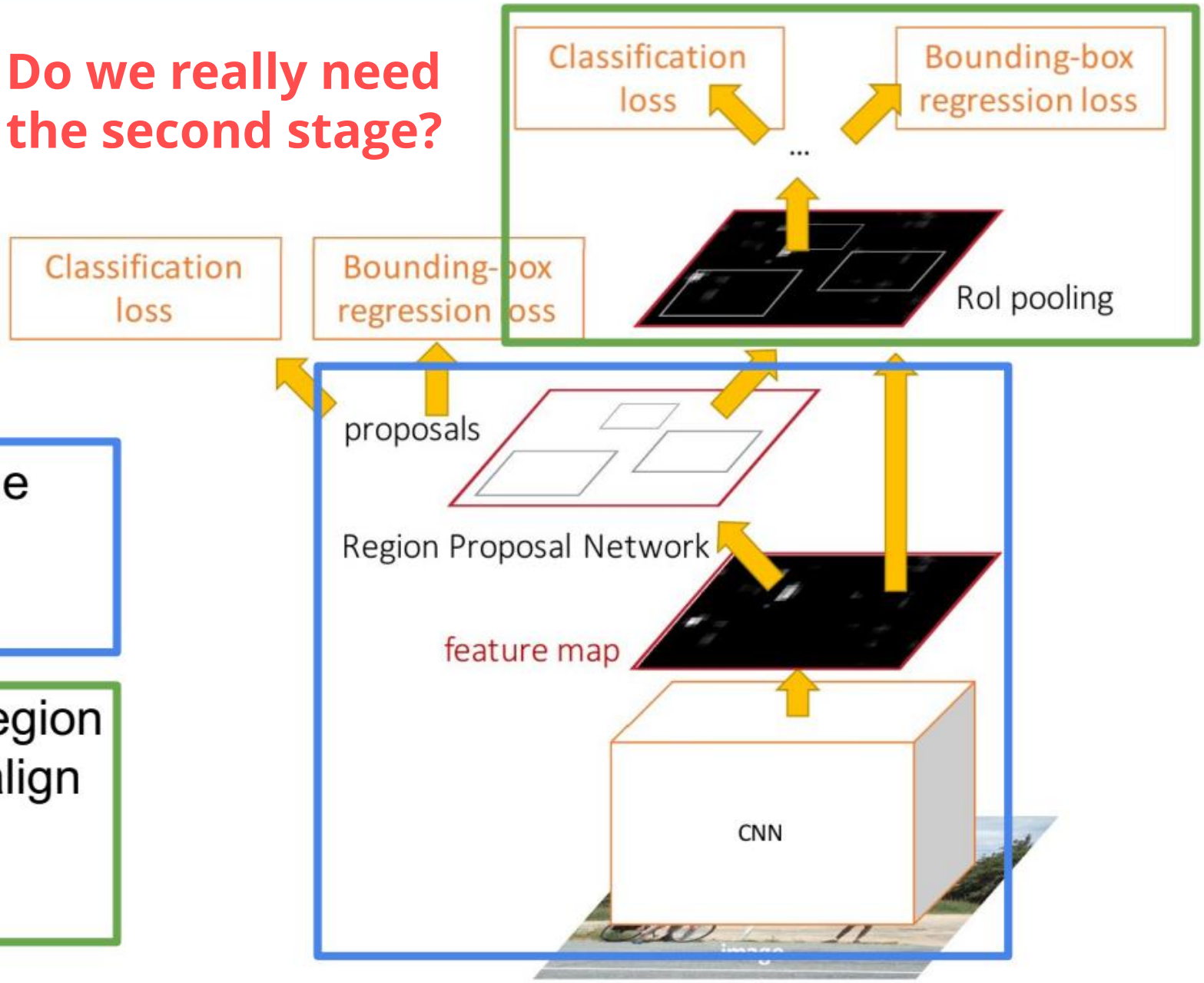
Make CNN do proposals!

Faster R-CNN is a
Two-stage object detector

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

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

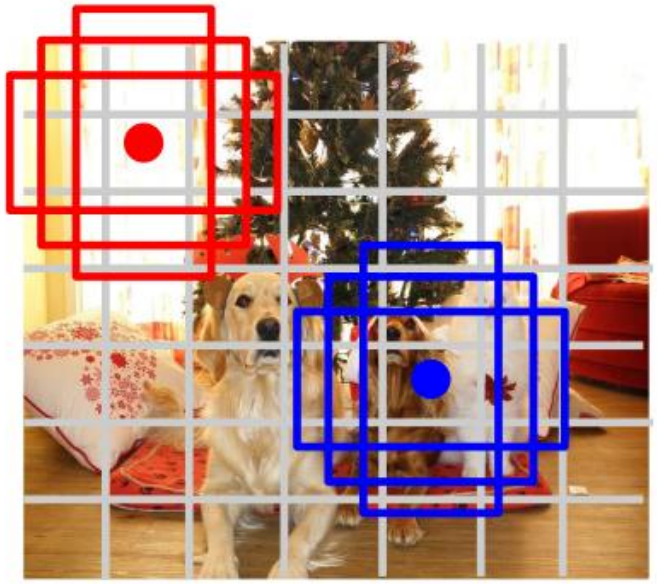
Do we really need the second stage?



Single-Stage ODs: YOLO / SSD / RetinaNet

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

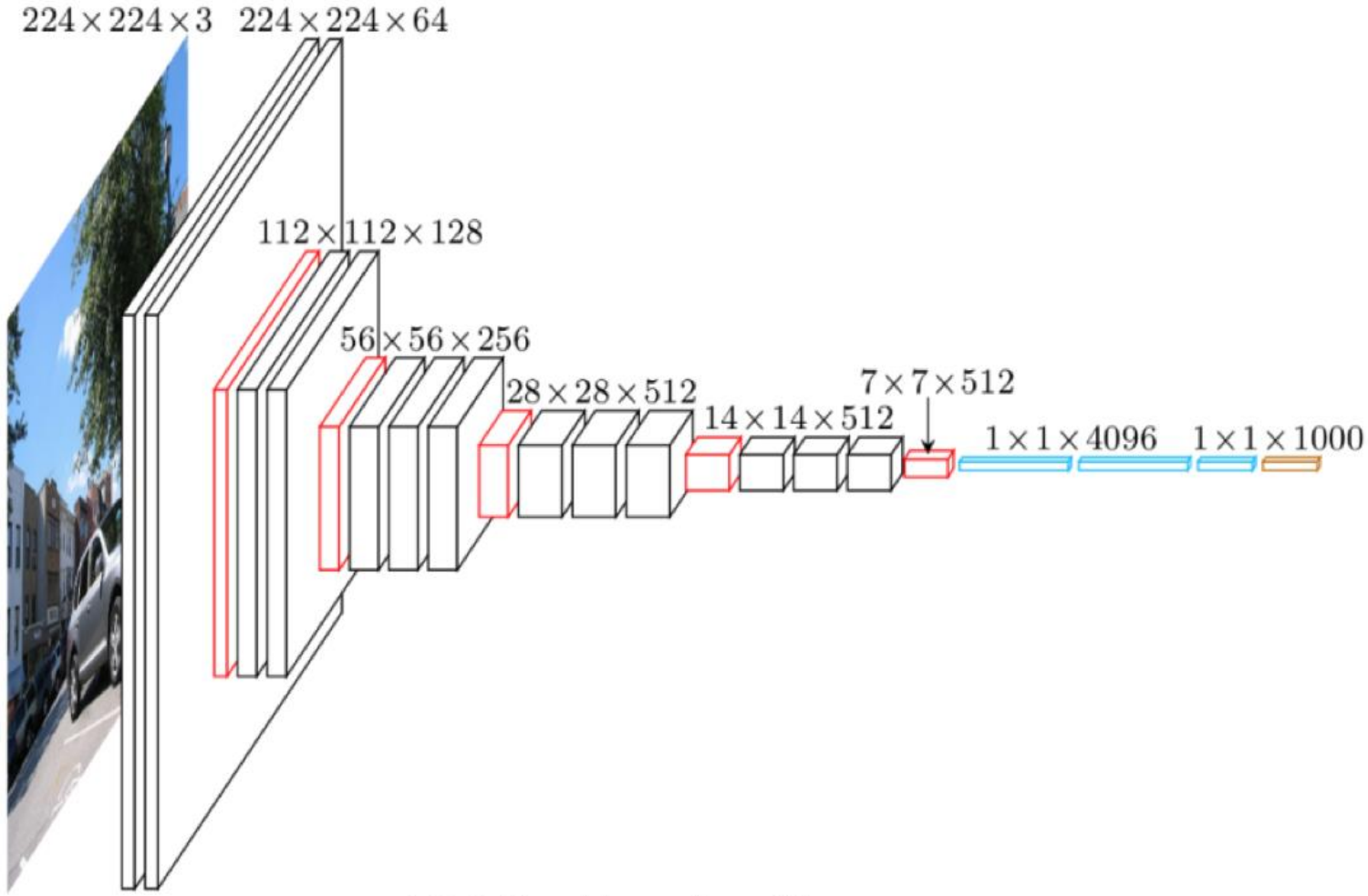
Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Single-Stage ODs: Predictions on a grid

1. We'll feed our input through a standard convolutional network to build a ***rich feature representation*** of the original image.

predict

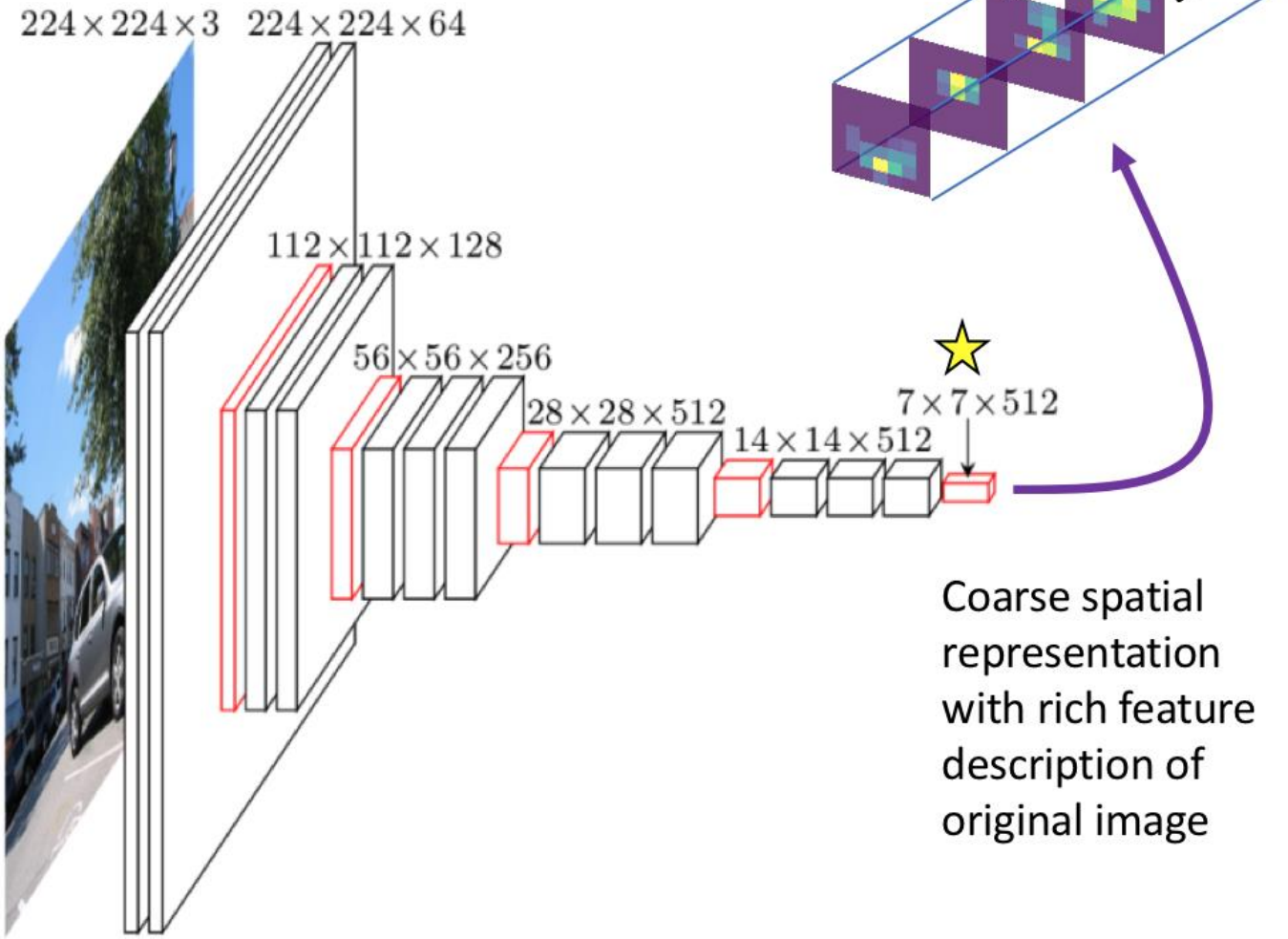


VGG “backbone” architecture

Single-Stage ODs: Predictions on a grid

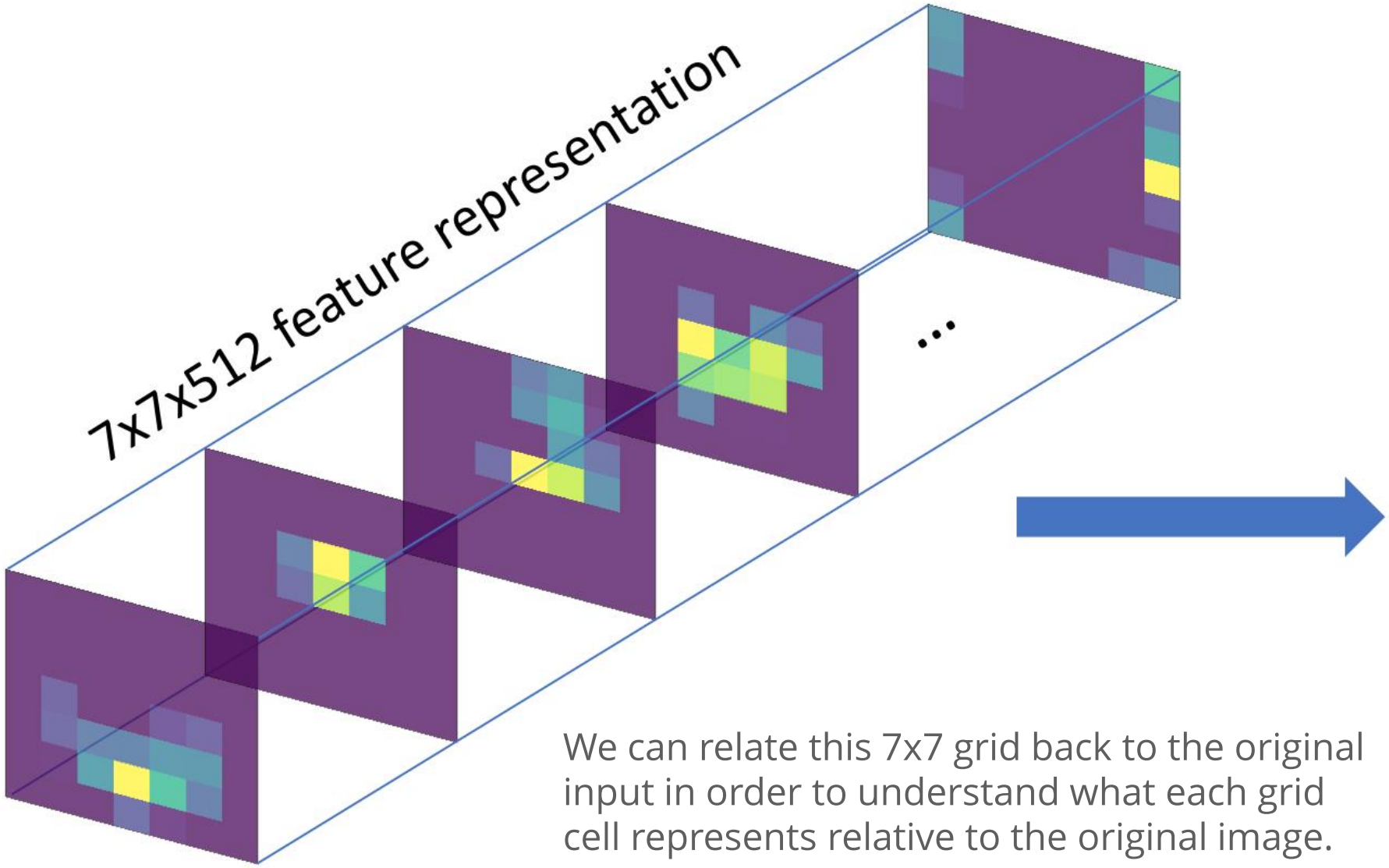
2. We'll then remove the last few layers of the network so that our backbone network outputs a collection of stacked feature maps which describe the original image in a ***low spatial resolution*** albeit a ***high feature (channel) resolution***.

predict

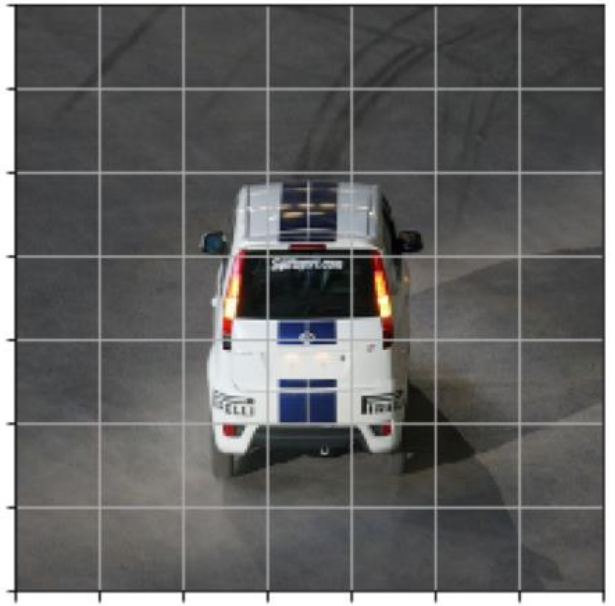


Coarse spatial representation with rich feature description of original image

Single-Stage ODs: Predictions on a grid



We can relate this 7x7 grid back to the original input in order to understand what each grid cell represents relative to the original image.



Visualizing the corresponding regions of each “pixel” in the 7x7 feature maps with the original 224x224 image

Single-Stage ODs: Predictions on a grid

We can also determine ***roughly*** where objects are located in the coarse (7x7) feature maps by observing which grid cell contains the center of our bounding box annotation.

We'll assign this grid cell as being ***"responsible"*** for detecting that specific object.

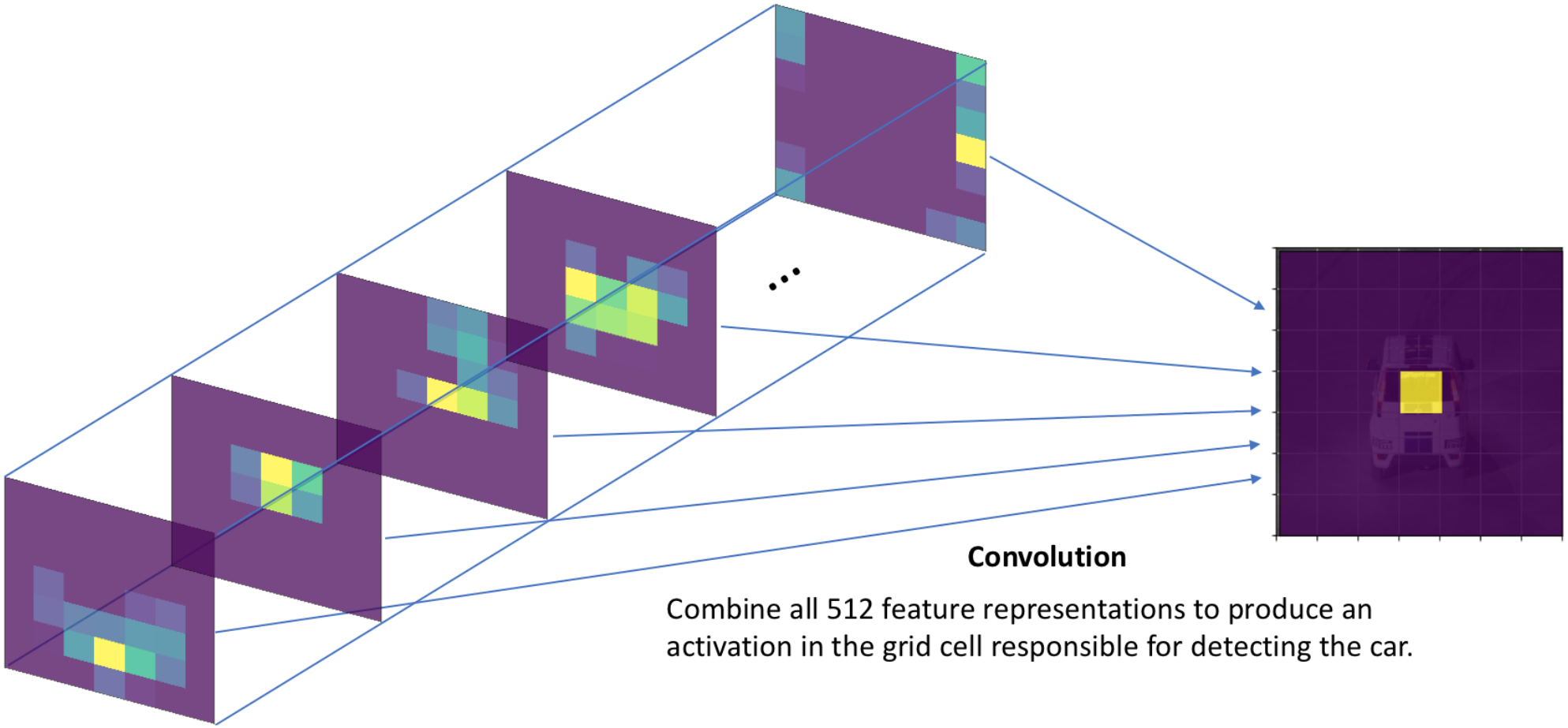
The input must have **labeled bounding boxes** for each object

We can plot the **center** of the bounding box, observing **which grid cell** it occurs in

We will make this grid cell **"responsible" for detecting** the car

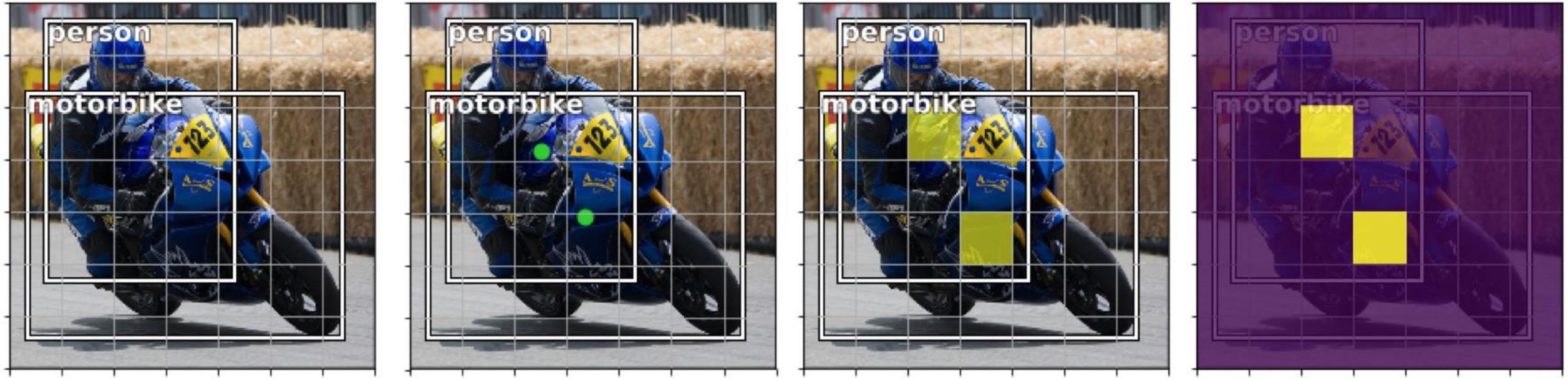
Single-Stage ODs: Predictions on a grid

In order to detect this object, we will add another convolutional layer and learn the kernel parameters which combine the context of all 512 feature maps in order to produce an activation corresponding with the grid cell which contains our object.



Single-Stage ODs: Predictions on a grid

If the input image contains multiple objects, we should have multiple activations on our grid denoting that an object is in each of the activated regions.



Single-Stage ODs: Predictions on a grid

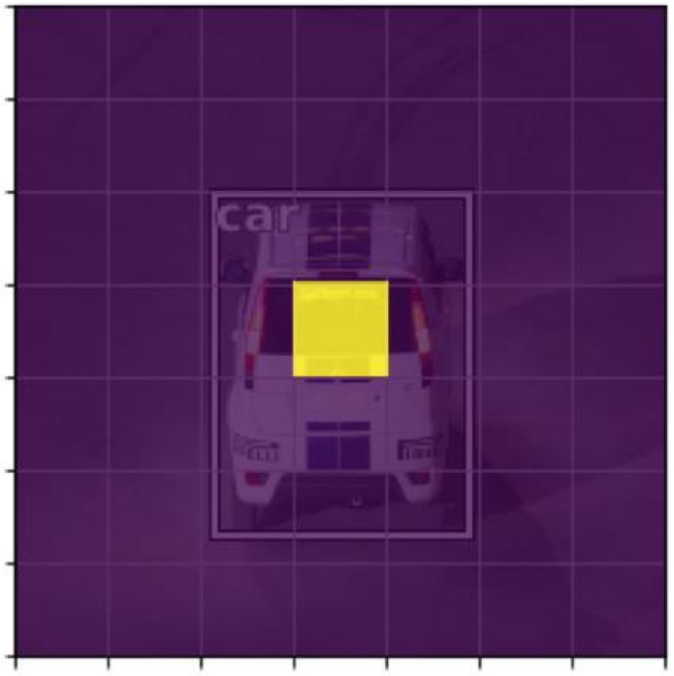
However, we cannot sufficiently describe each object with a single activation. In order to fully describe a detected object, we'll need to define:

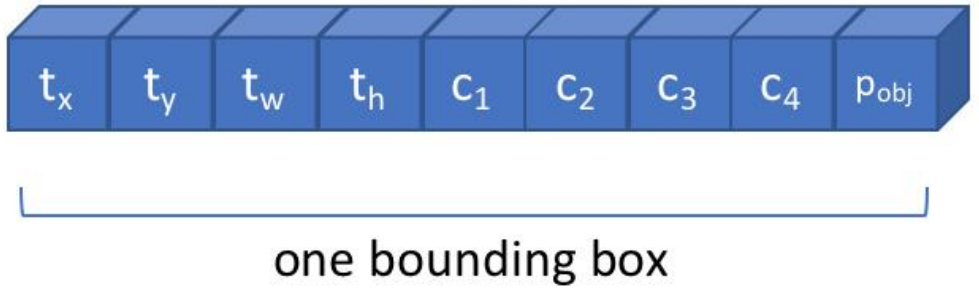
- The likelihood that a grid cell contains an object ($\boldsymbol{p_{obj}}$)
- Which class the object belongs to ($\boldsymbol{c_1, c_2, \ldots, c_C}$)
- Four bounding box descriptors to describe the x coordinate, y coordinate, width, and height of a labeled box ($\boldsymbol{t_x, t_y, t_w, t_h}$)

Thus, we'll need to learn a convolution filter for each of the above attributes such that we produce $\boldsymbol{5 + C}$ output channels to describe a single bounding box at each grid cell location.

Single-Stage ODs: Predictions on a grid

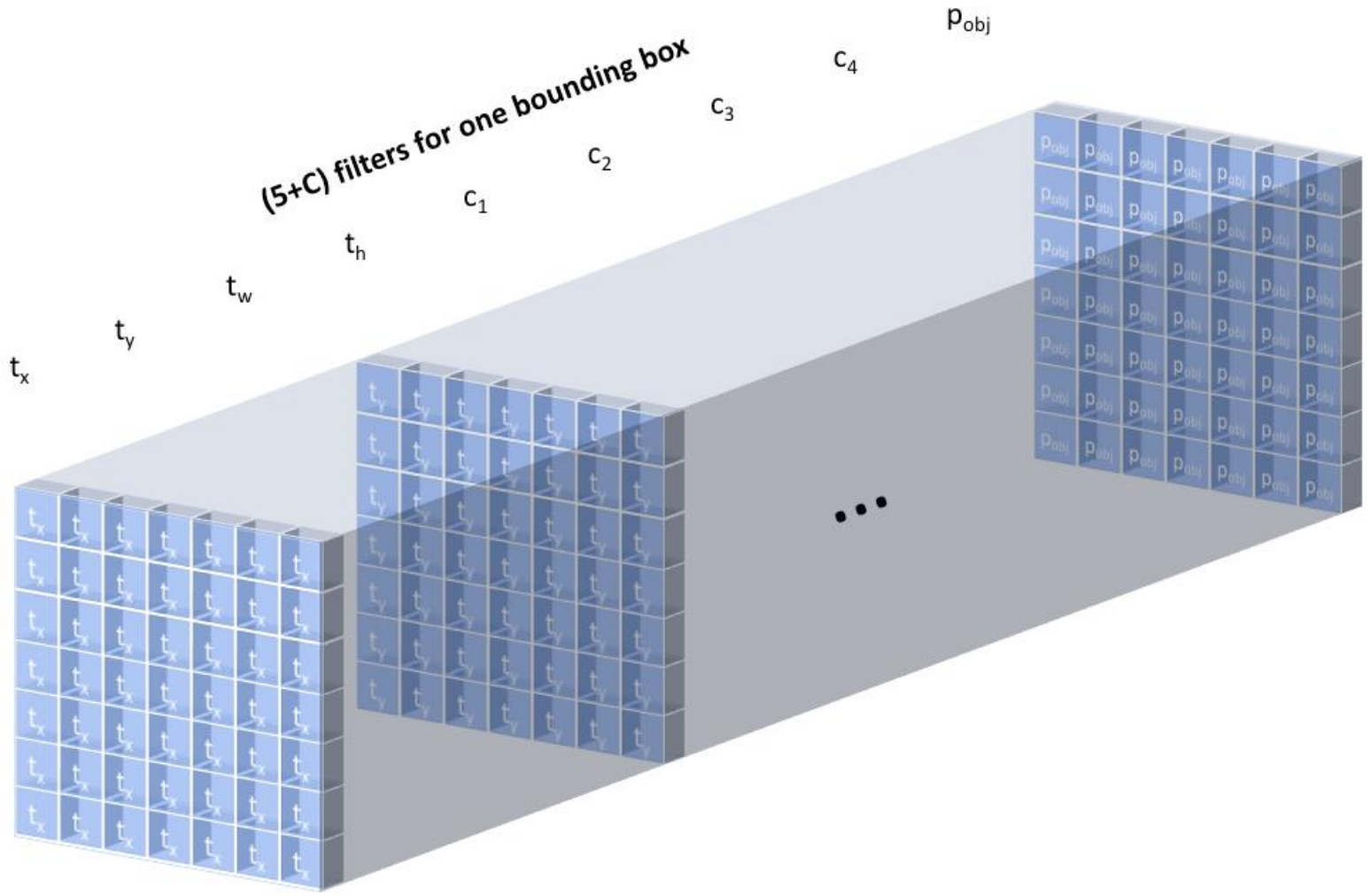
This means that we'll learn a set of weights to look across all 512 feature maps and determine which grid cells are likely to contain an object, what classes are likely to be present in each grid cell, and how to describe the bounding box for possible objects in each grid cell.





Single-Stage ODs: Predictions on a grid

The full output of applying $5 + C$ convolutional filters is shown below for clarity, producing one bounding box descriptor for each grid cell.



Single-Stage ODs: Predictions on a grid

However, some images might have ***multiple objects*** which "belong" to the same grid cell. We can alter our layer to produce $\boldsymbol{B}(\mathbf{5}+\boldsymbol{C})$ filters such that we can predict $\boldsymbol{B}$ bounding boxes for each grid cell location.

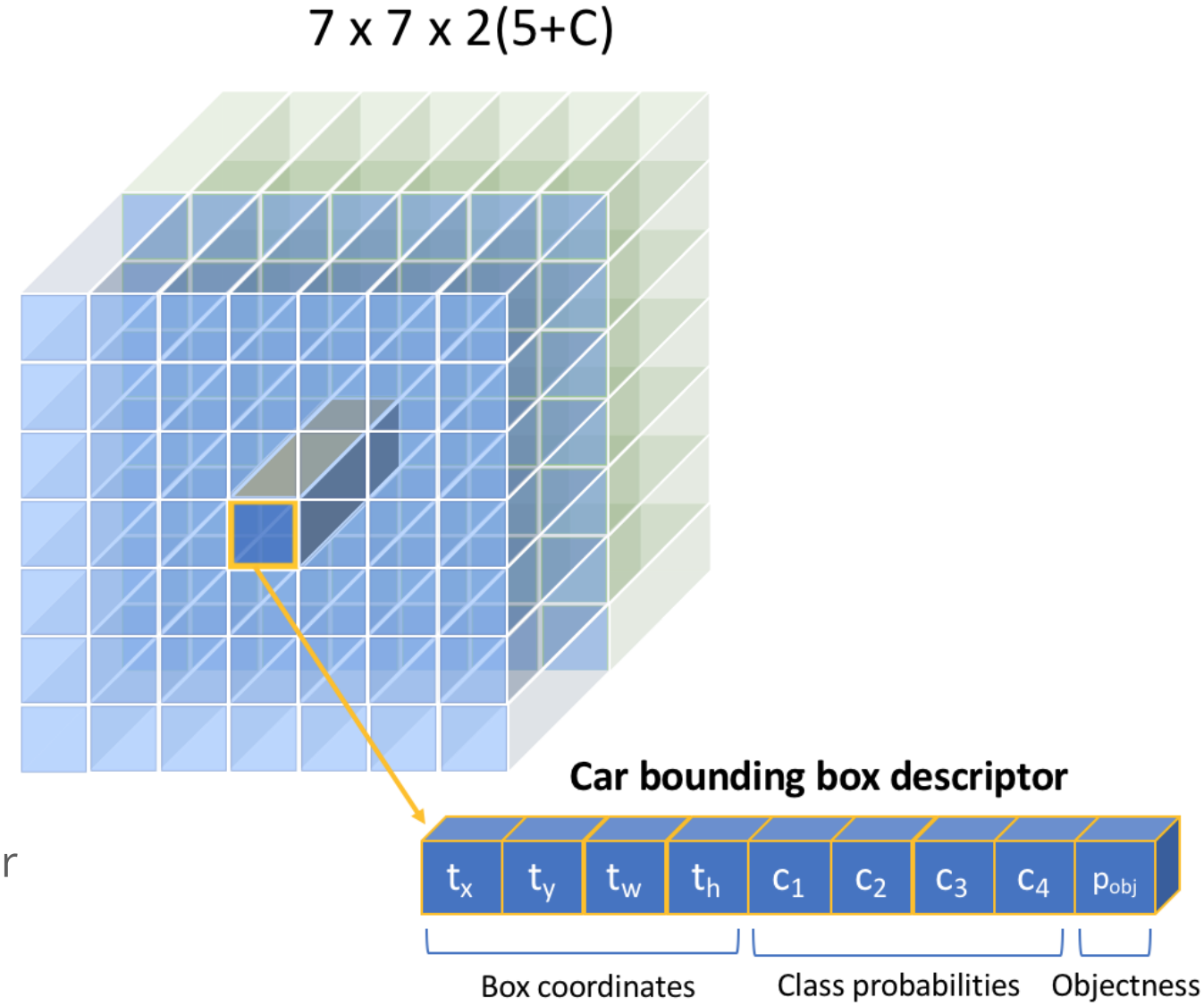
Learning 2 bounding boxes for each grid cell requires 18 channels
(assuming 4 possible classes)

one bounding box

Single-Stage ODs: Predictions on a grid

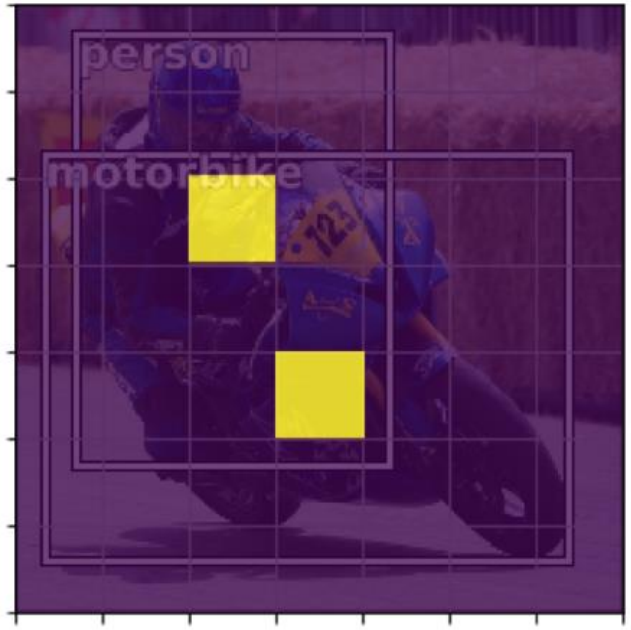
we can see that our model will always produce a fixed number of $N \times N \times B$ predictions for a given image.

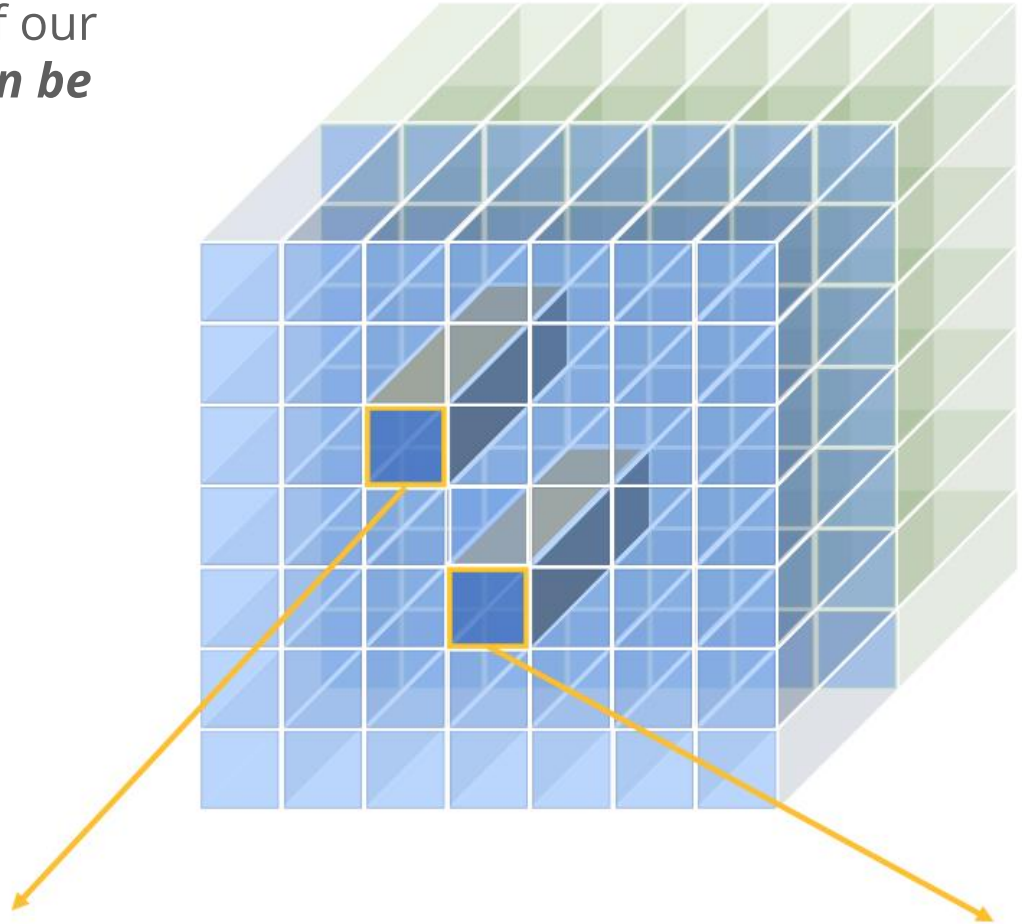
We can then filter our predictions to only consider bounding boxes which has a p_{obj} above some defined threshold.



Single-Stage ODs: Predictions on a grid

Because of the convolutional nature of our detection process, ***multiple objects can be detected in parallel***.





Person bounding box descriptor

Motorbike bounding box descriptor

Single-Stage ODs: Predictions on a grid

The ***"predictions on a grid"*** approach produces a ***fixed*** number of bounding box predictions for each image. However, we would like to filter these predictions in order to only output bounding boxes for objects that are actually likely to be in the image.

Moreover, we want a ***single bounding box*** prediction for each object detected.

We can filter out most of the bounding box predictions by only considering predictions with a p_{obj} above some defined confidence ***threshold***. However, we still may be left with multiple high-confidence predictions describing the same object.

Thus, we need a method for removing redundant object predictions such that each object is described by a single bounding box.

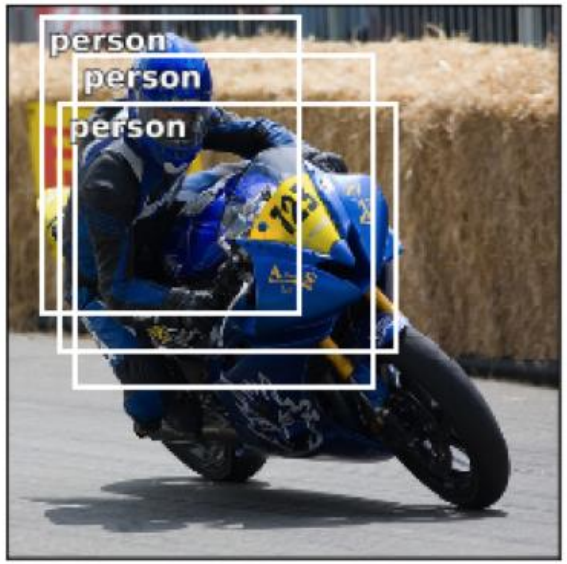
To accomplish this, we'll use a technique known as ***non-max suppression***.

Non-Max Suppression

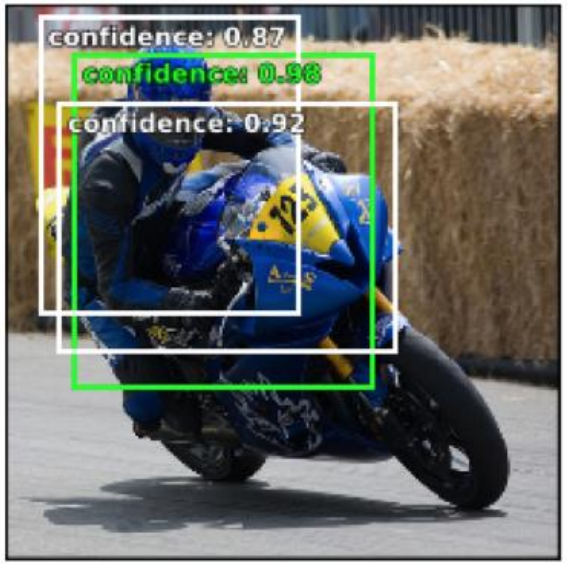
We'll perform non-max suppression on **each class separately**.

For each class...

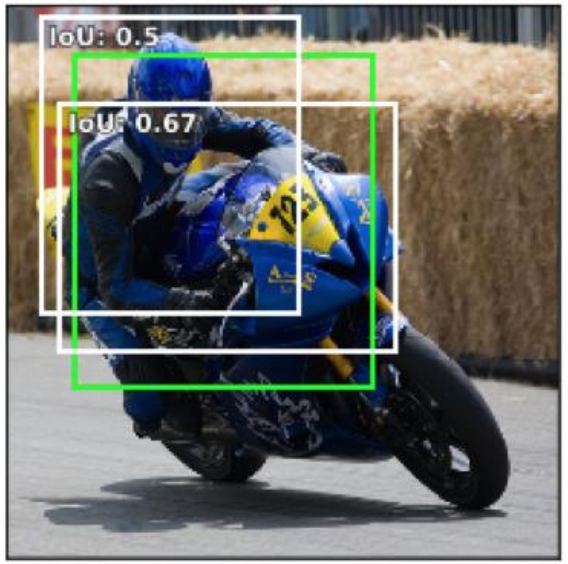
Repeat with next highest confidence prediction until no more boxes are being suppressed



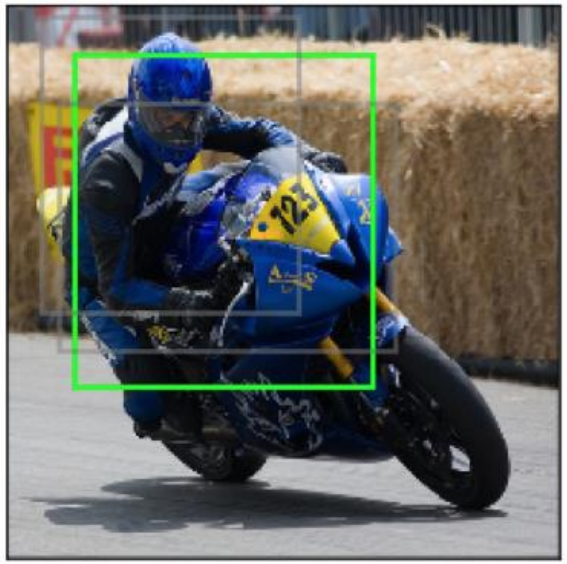
After filtering out low confidence predictions, we may still be left with ***redundant*** **detections**



Select the bounding box prediction with the **highest confidence**



Calculate the IoU between the selected box and all remaining predictions

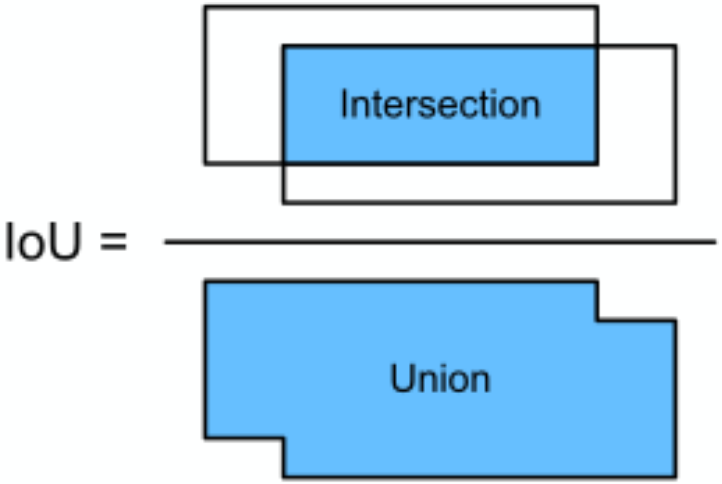


Remove any boxes which have an IoU score above some defined threshold

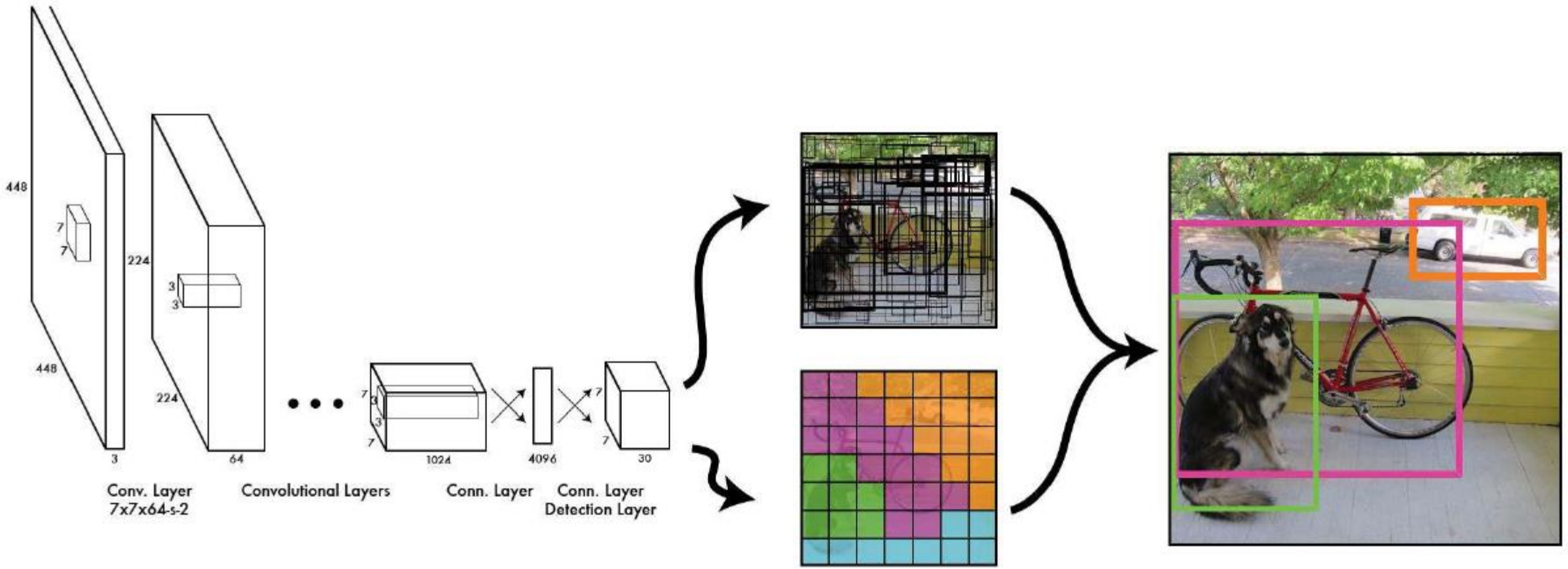
Intersection over Union (IoU)

The range of an IoU is between $\mathbf{0}$ and $\mathbf{1}$:

$\mathbf{0}$ means that two bounding boxes do not overlap at all, while $\mathbf{1}$ indicates that the two bounding boxes are equal.

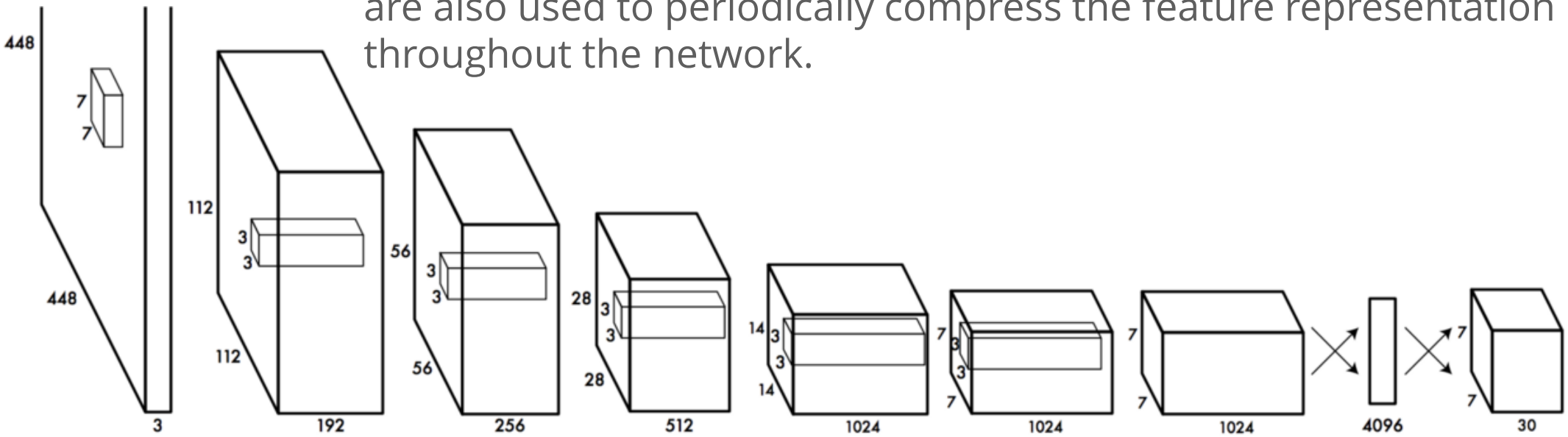


You Only Look Once: YOLO



YOLO: Backbone Network

The original YOLO used a modified ***GoogLeNet*** as the backbone network. Later ***DarkNet-19*** is used, which follows the general design of a $\mathbf{3 \times 3}$ filters, doubling the number of channels at each pooling step; $\mathbf{1 \times 1}$ filters are also used to periodically compress the feature representation throughout the network.



YOLO: Backbone Network

Yolov4 paper introduces a new, larger model named ***DarkNet-53*** which offers improved performance over its predecessor.

All of these models were first pre-trained as image classifiers before being adapted for the detection task.

All of these models were first pre-trained as image classifiers before being adapted for the detection task. In the second iteration of the YOLO model.

Adapting the classification network for detection simply consists of removing the last few layers of the network and adding a convolutional layer with $\boldsymbol{B}(\mathbf{5}+\boldsymbol{C})$ filters to produce the $\boldsymbol{N} \times \boldsymbol{N} \times \boldsymbol{B}$ bounding box predictions.

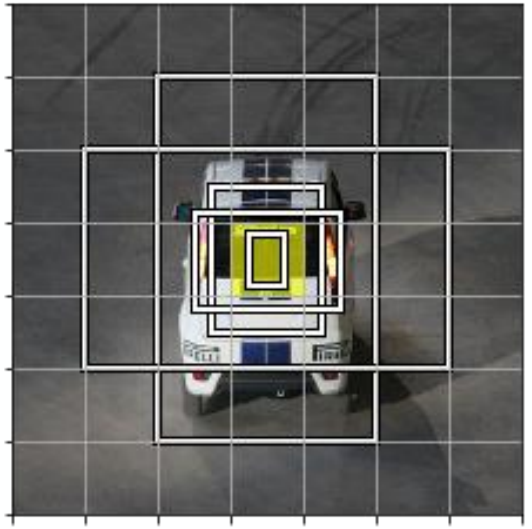
YOLO: Concept of Anchor Boxes

Rather than expecting the model to directly produce ***unique bounding box descriptors*** for each new image, we will ***define a collection of bounding boxes with varying aspect ratios*** which embed some prior information about the shape of objects we're expecting to detect.

Redmond offers an approach towards discovering the best aspect ratios by doing k-means clustering (with a custom distance metric) on all of the bounding boxes in your training dataset.

In the image below, you can see a collection of 5 bounding box priors (aka ***anchor boxes***) for the grid cell highlighted in yellow.

With this formulation, each of the B bounding boxes explicitly specialize in detecting objects of a specific size and aspect ratio.

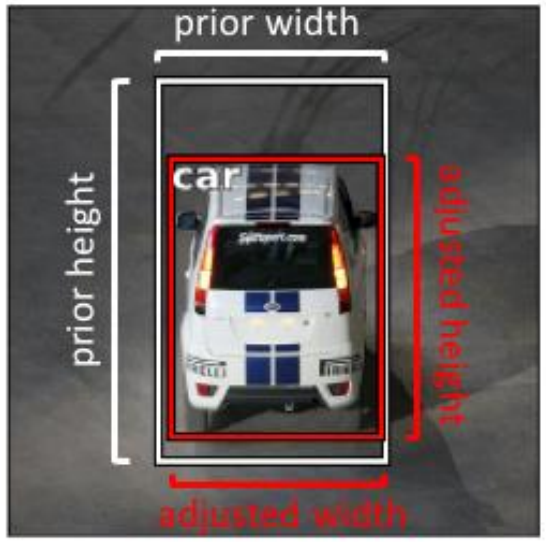


Note: These anchor boxes are present for each cell in our prediction grid.

YOLO: Concept of Anchor Boxes

Rather than directly predicting the bounding box dimensions, we'll reformulate our task in order to simply predict the offset from our bounding box prior dimensions such that we can fine-tune our predicted bounding box dimensions.

This reformulation makes the prediction task easier to learn.



YOLO: Objectness (assigning labeled objects to a bounding box)

In the first version of the model, the "***objectness***" score p_{obj} was trained to approximate the Intersection over Union (***IoU***) between the predicted box and the ground truth label.

When we calculate our loss during training, we'll match objects to whichever bounding box prediction (on the same grid cell) has the ***highest IoU score***.

After the addition bounding box priors in YOLOv2, we can simply assign labeled objects to whichever ***anchor box*** (on the same grid cell) has the ***highest IoU*** score with the labeled object.

In the third version, "***objectness***" target score p_{obj} was defined to be $\mathbf{1}$ for the bounding boxes with ***highest IoU*** score for each given target, and ***0*** for all remaining boxes.

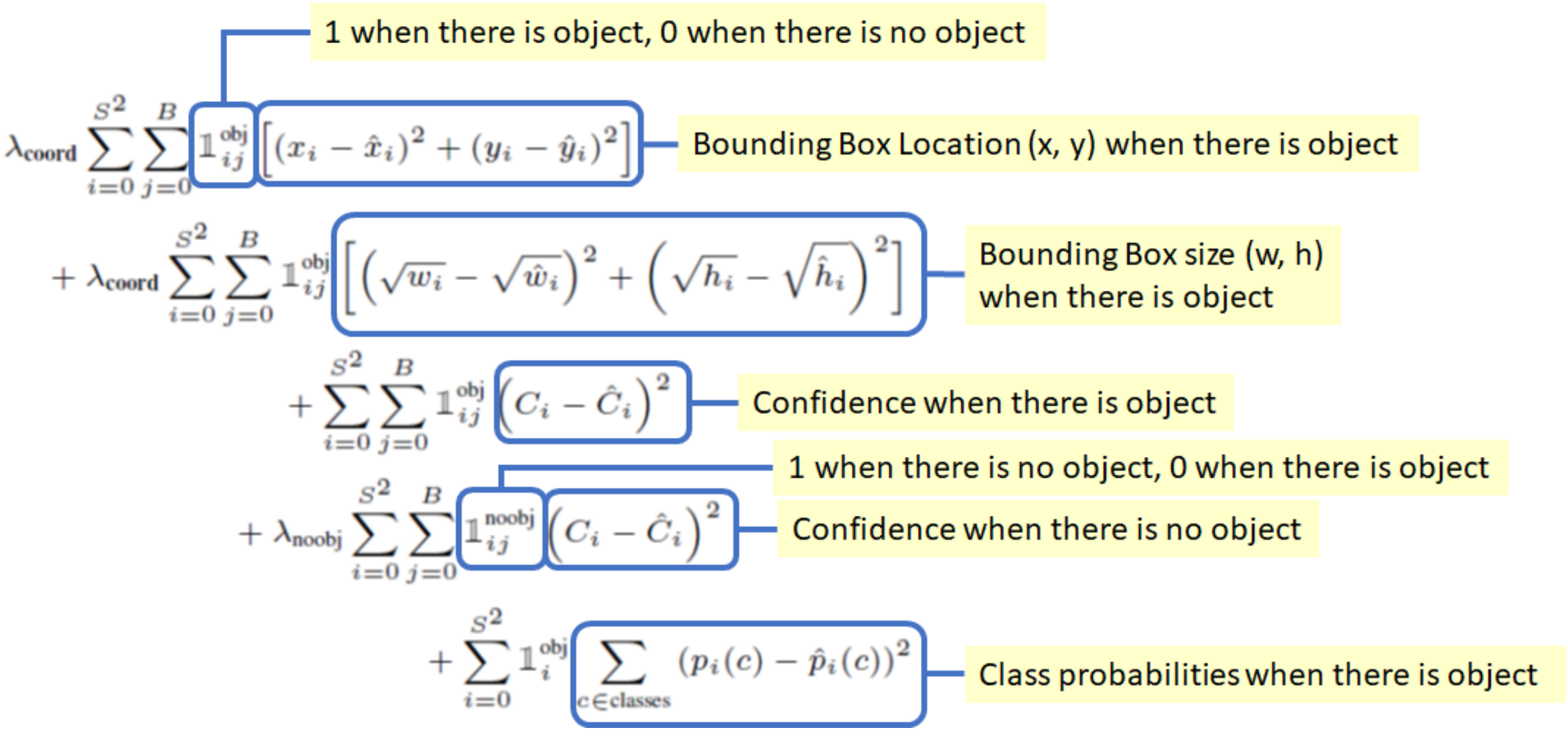
However, we will not include bounding boxes which have a high IoU score (above some threshold) but not the highest score when calculating the loss.

YOLO: Class Labels

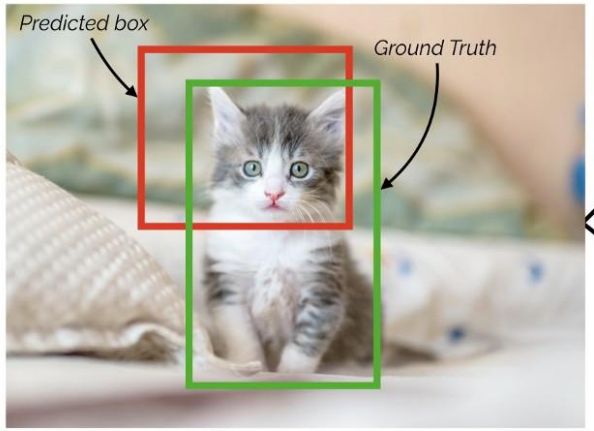
Originally, class prediction was performed at the grid cell level. This means that a single grid cell could not predict multiple bounding boxes of different classes.

This was later revised to predict class for each bounding box using a ***softmax*** activation across classes and a ***cross entropy loss***.

YOLO: Loss Function



YOLO: Mean Average Precision



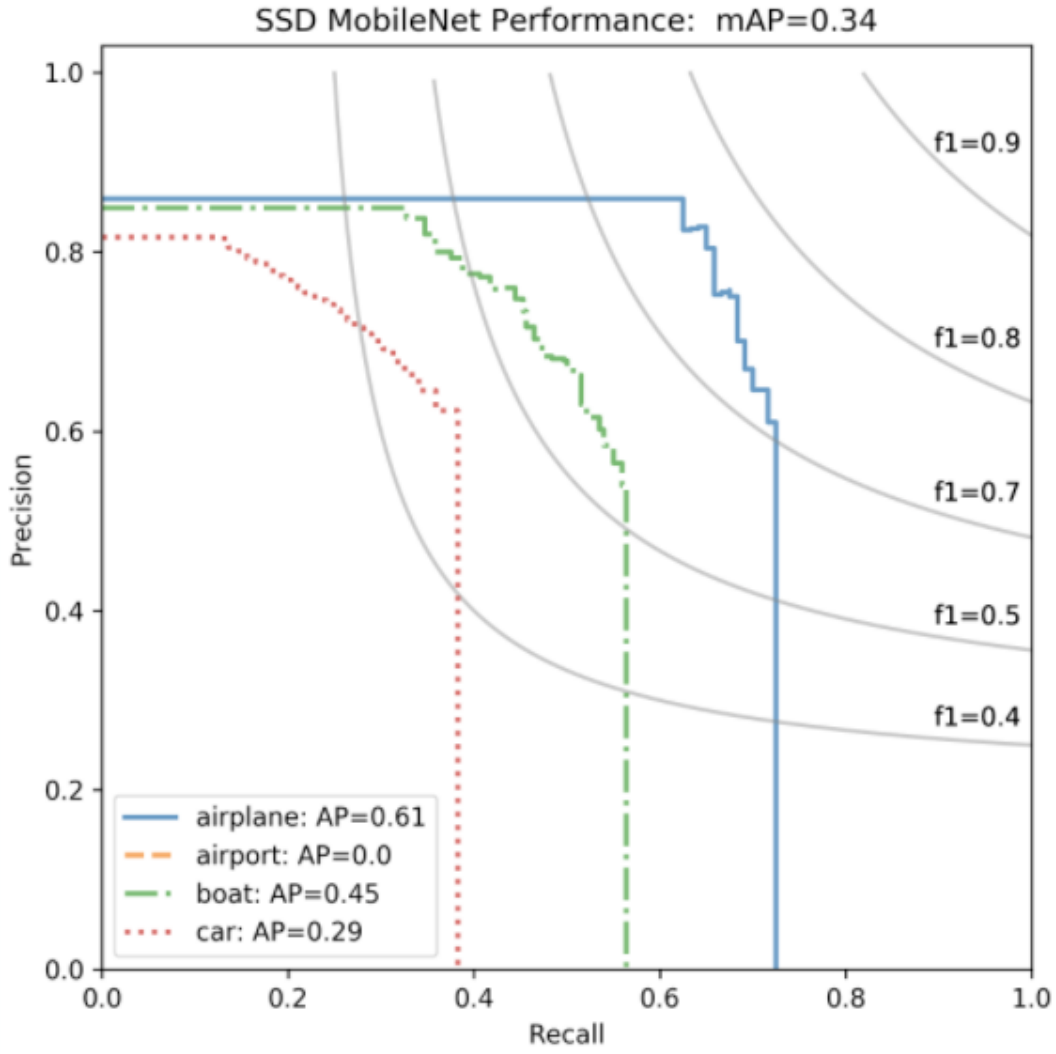
IoU threshold = 0.5 = *False Positive (FP)*

IoU threshold = 0.2 = *True Positive (TP)*

IoU for the prediction = ~0.3

$$\textbf{Precision} = \frac{TP}{TP + FP} \qquad \textbf{Recall} = \frac{TP}{TP + FN}$$

The mean Average Precision or mAP score is calculated by taking the mean AP (area under the precision-recall curve) over all classes and/or overall IoU thresholds, depending on different detection challenges that exist.



YOLO: Further Reading

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

https://github.com/AlexeyAB/darknet

https://arxiv.org/abs/2004.10934

https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/

https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/

https://sannaperzon.medium.com/yolov3-implementation-with-training-setup-from-scratch-30ecb9751cb0

https://docs.ultralytics.com/

Thank You!