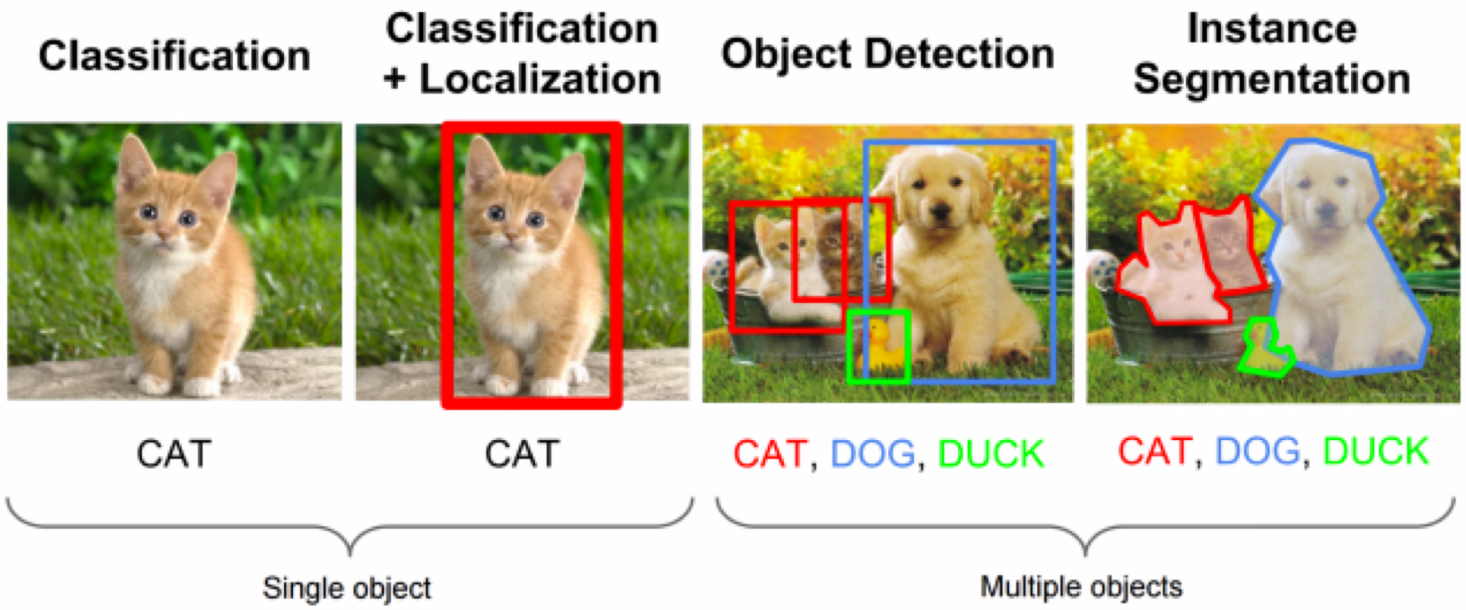
You Only Look Once
YOLO

ABNANE Ibtissam

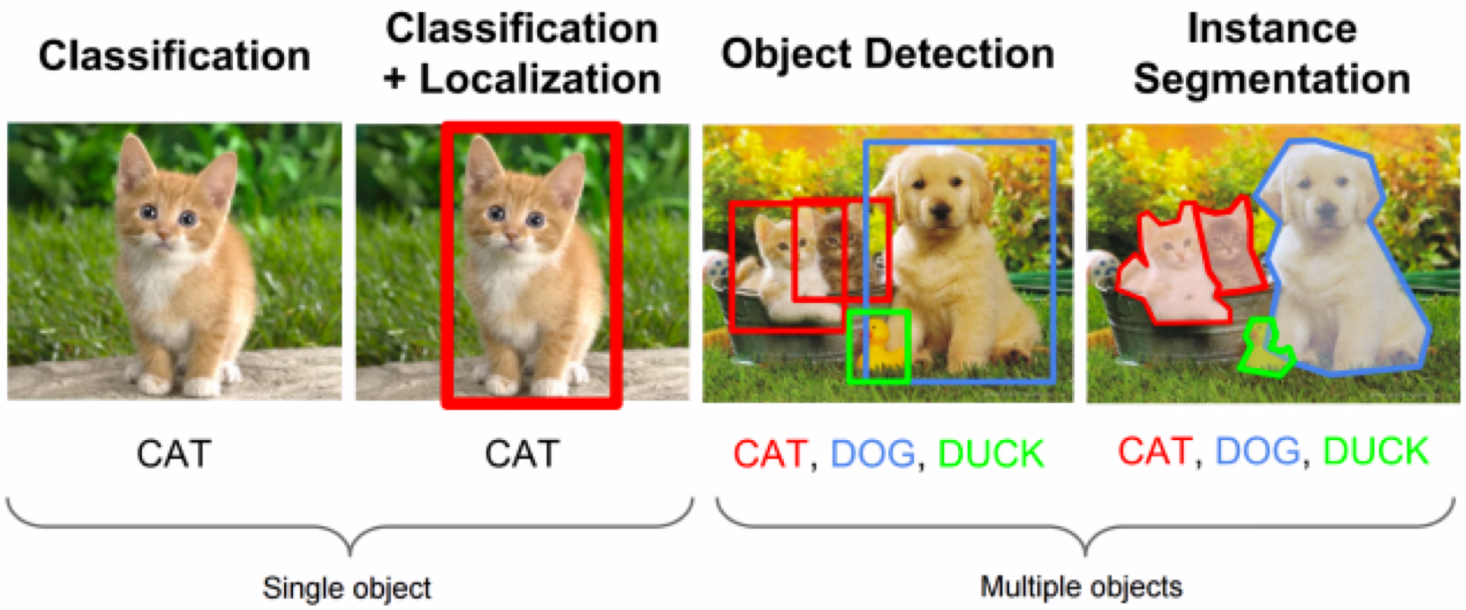
Computer vision

Computer vision is a multidisciplinary field of artificial intelligence and computer science that focuses on enabling computers to interpret and understand visual information from the world, much like a human visual system.



Computer vision

Image Classification: Assigning a label or category to an entire image. For example, identifying whether an image contains a cat, dog, car, or any other object or scene.



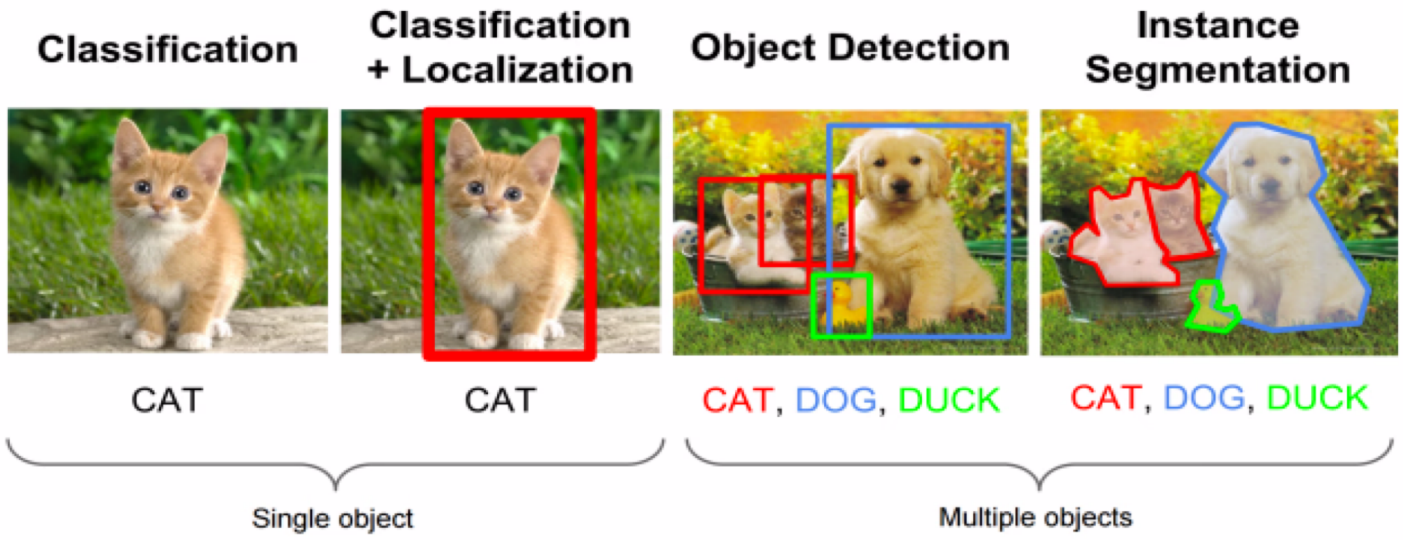
Computer vision

Classification:

- the process of assigning an object or region in an image to one or more predefined categories or classes.
- answers the question: "What is in the image? »
- The output is a class label or a probability distribution over possible classes.

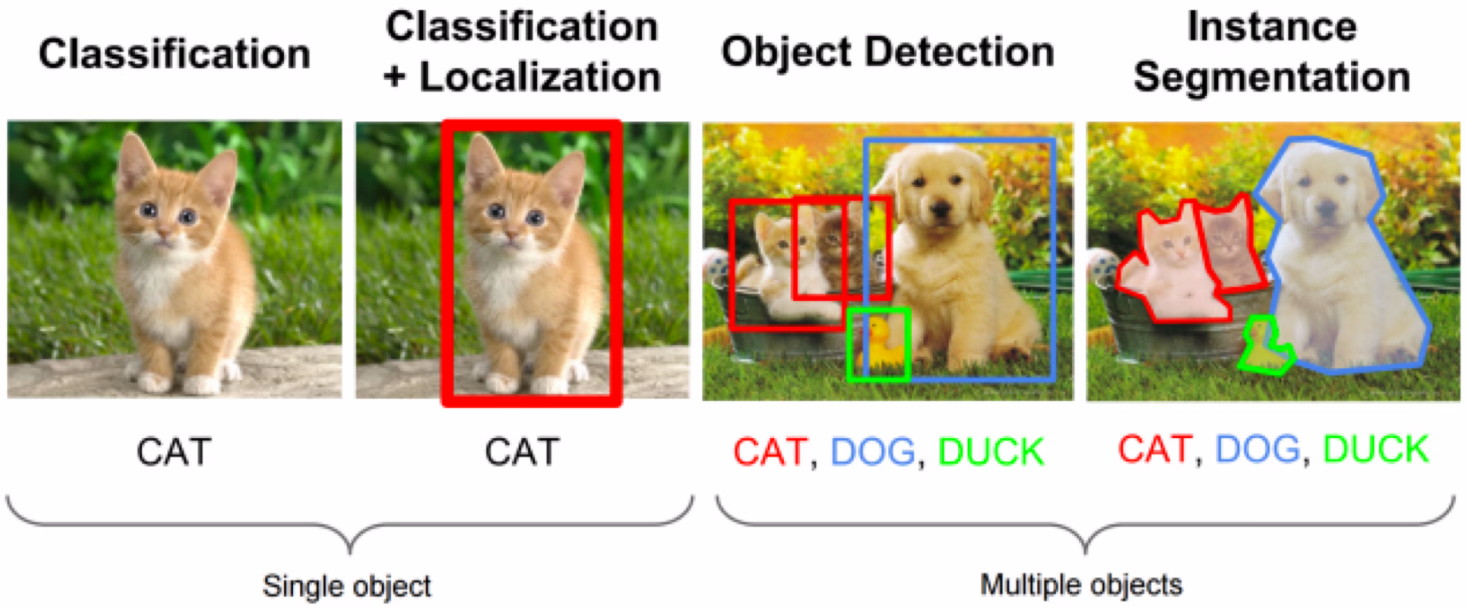
Localization:

- the task of not only identifying objects within an image but also determining their precise location by drawing bounding boxes around them.
- answers the question: "Where are the objects in the image? »
- identify the object (e.g., a cat or a dog) and provide the coordinates that define the rectangular bounding boxes around it.



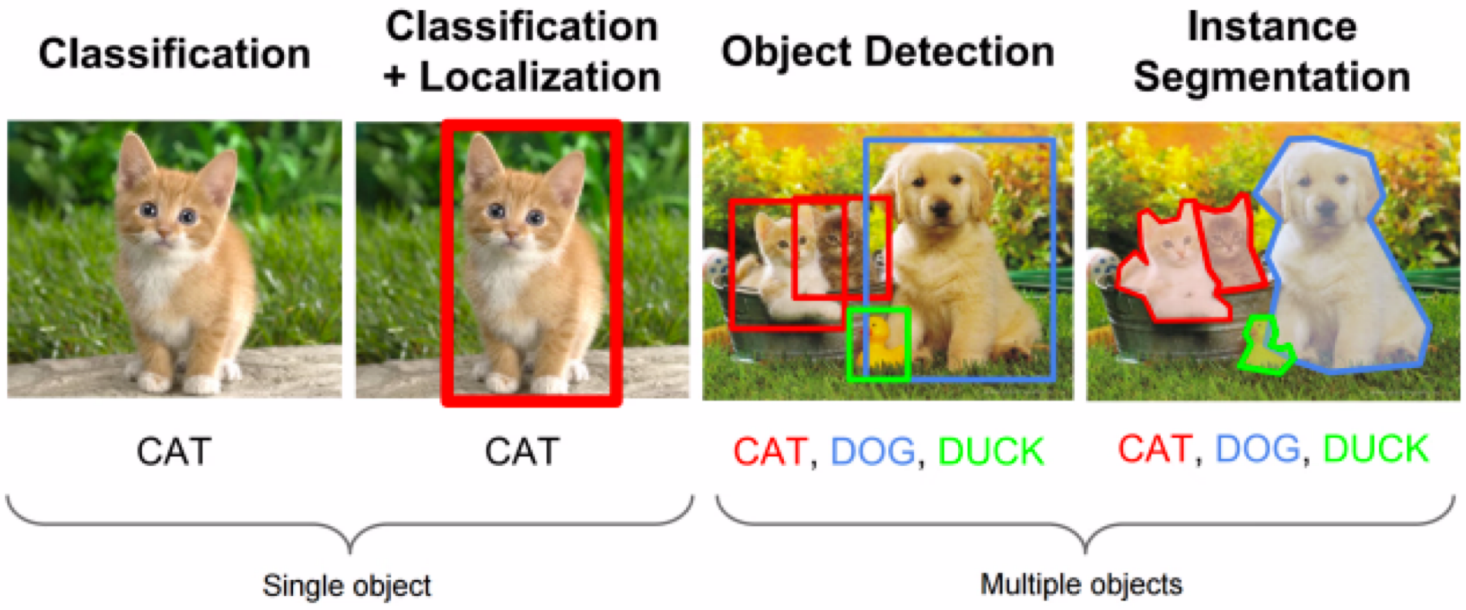
Computer vision

Object Detection: Locating and identifying objects or regions of interest within an image or video stream. Object detection algorithms can draw bounding boxes around objects, classify them, and track their movement.



Computer vision

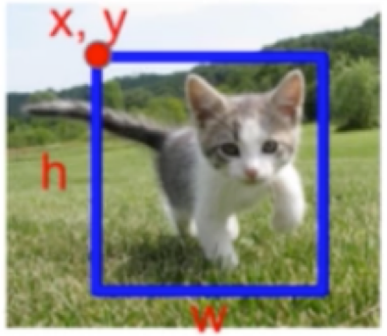
Semantic Segmentation: In addition to object classification and localization, instance segmentation further segments each object into pixels that belong to it. This means that it identifies the exact boundaries and outlines of each object in the image, allowing for pixel-level understanding.



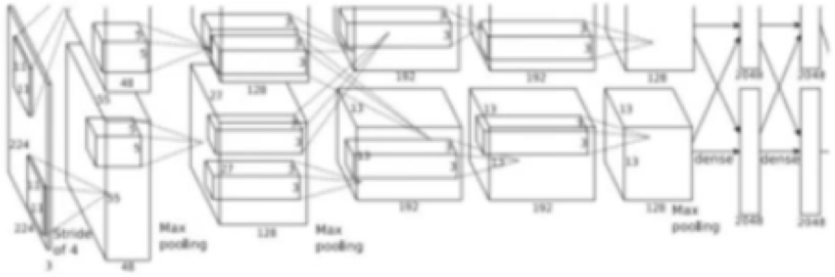
Computer Vision

- **Image classification:** We want to just assign a category label to the input image.
- **Classification plus localization:** In addition to predicting what the category is, you also want to know where is that object in the image?
- **Object detection:** Given a fixed set of categories, every time one of those categories appears in the input image, we want to draw a box around it and we want to predict the category of that box.

Classification+ Localization



This image is CC0 public domain



Fully Connected: 4096 to 1000

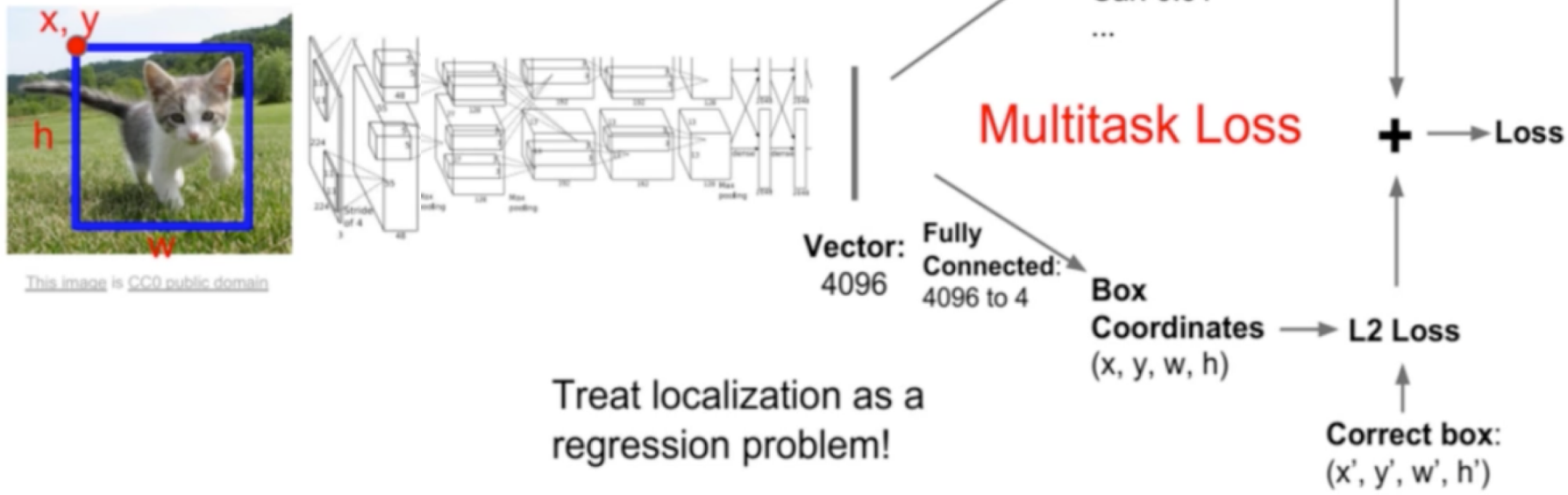
Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Vector: 4096

Fully Connected: 4096 to 4

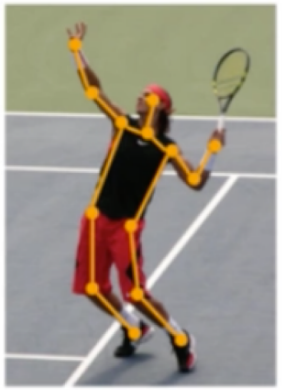
Box Coordinates
(x, y, w, h)

Classification+ Localization



Classification+ Localization

Example: Human pose estimation

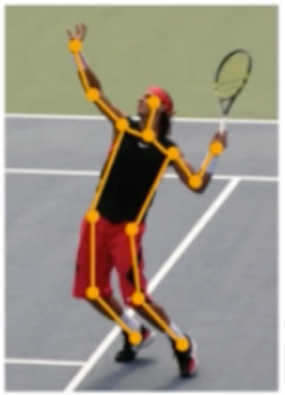


Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

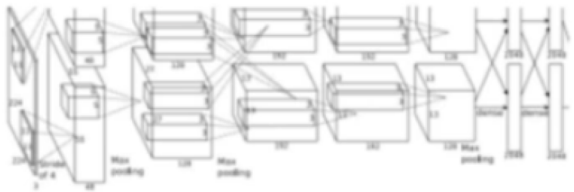
Classification+ Localization

Example: Human pose estimation



Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top



Vector: 4096

Left foot: (x, y) → L2 loss

Right foot: (x, y) → L2 loss

...

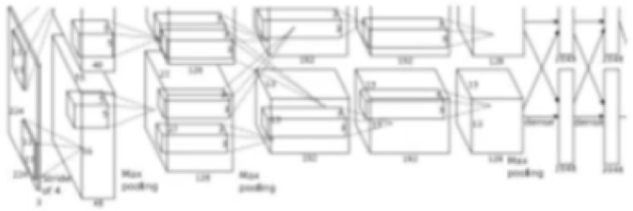
Head top: (x, y) → L2 loss

Correct left foot: (x', y')

Correct head top: (x', y')

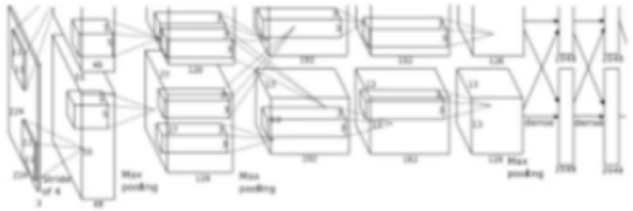
+ → Loss

Object Detection: multiple objects



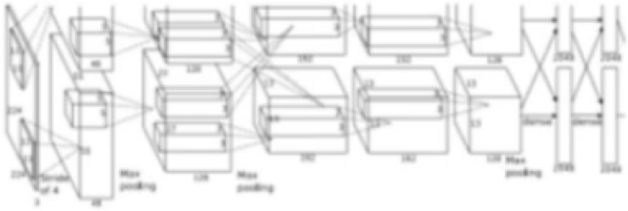
CAT: (x, y, w, h)

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)
....

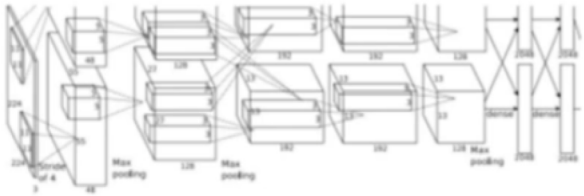
Many numbers!

Each image needs a different number of outputs!

Probem: the length of the output layer is not constant

Object Detection: multiple objects

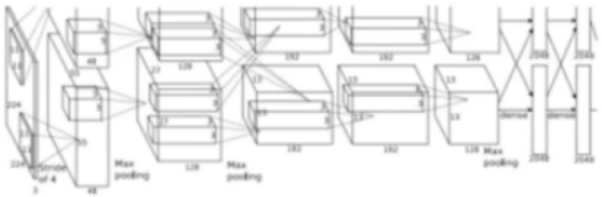
Apply a CNN to different crops (regions) of the image => Sliding window
CNN classifies each region as: object or background



Dog? NO
Cat? NO
Background? YES

Object Detection: multiple objects

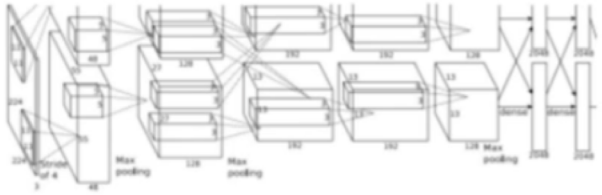
Apply a CNN to different crops (regions) of the image => Sliding window
CNN classifies each region as: object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: multiple objects

Apply a CNN to different crops (regions) of the image => Sliding window
CNN classifies each region as: object or background

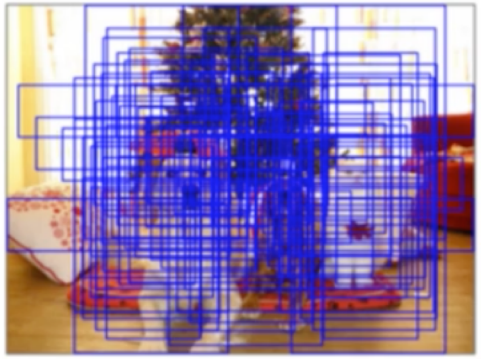


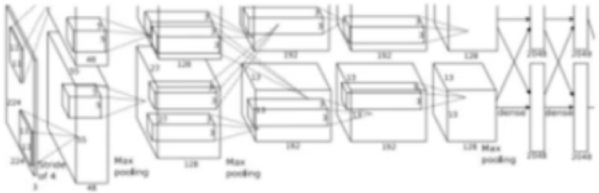
Dog? YES
Cat? NO
Background? NO

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

Object Detection: multiple objects

Apply a CNN to different crops (regions) of the image => Sliding window
CNN classifies each region as: object or background





Dog? YES
Cat? NO
Background? NO

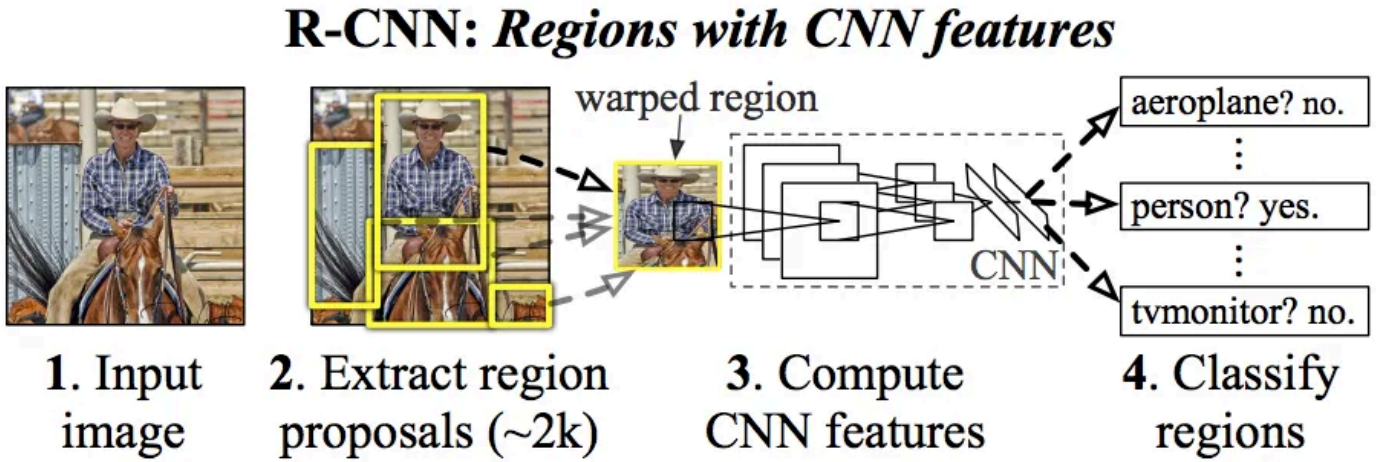
Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

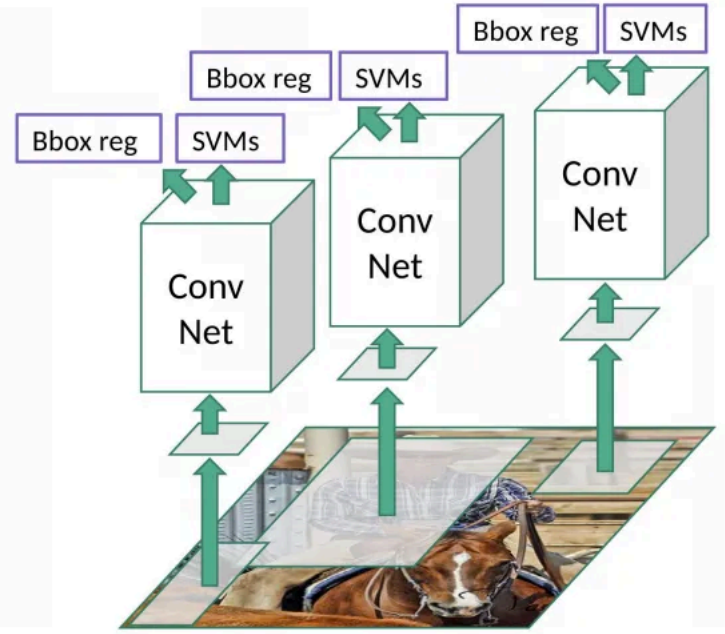
R-CNN

Is there a method to propose regions?
⇒ Region proposal technique
⇒ 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

R-CNN



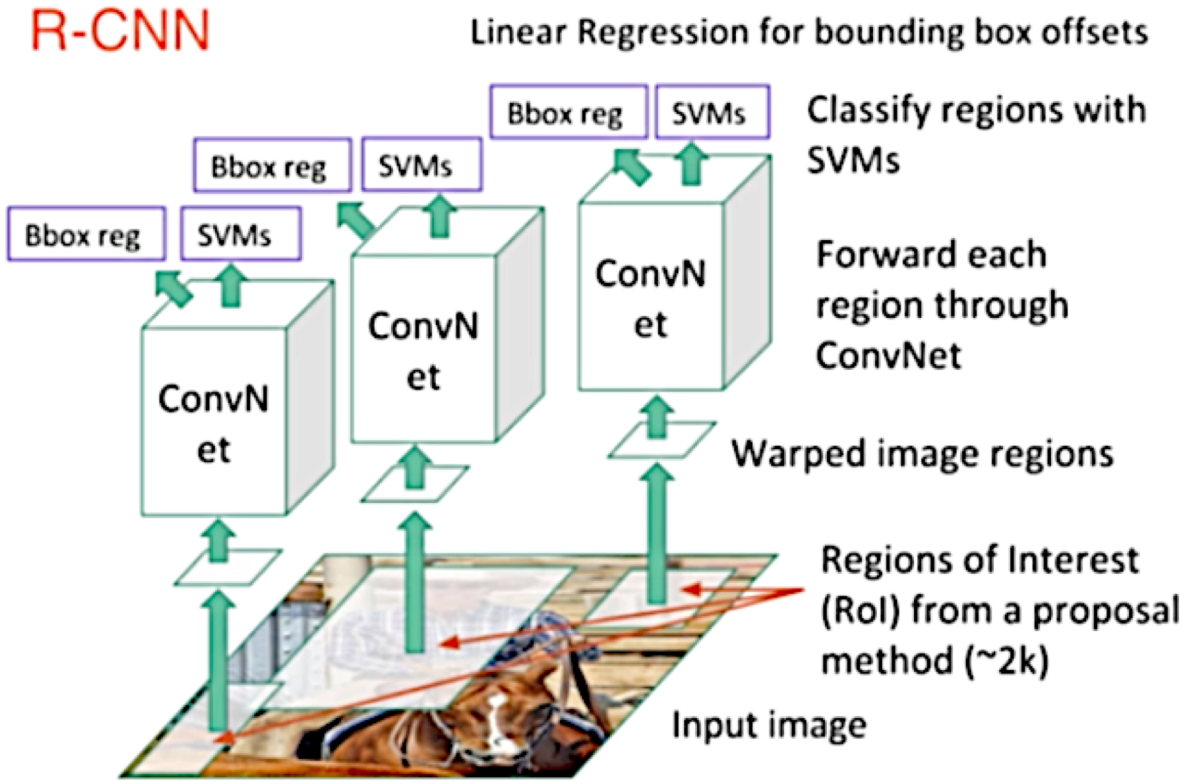


The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal.

R-CNN

R-CNN, or Region-based CNN, consisted of 4 simple steps:

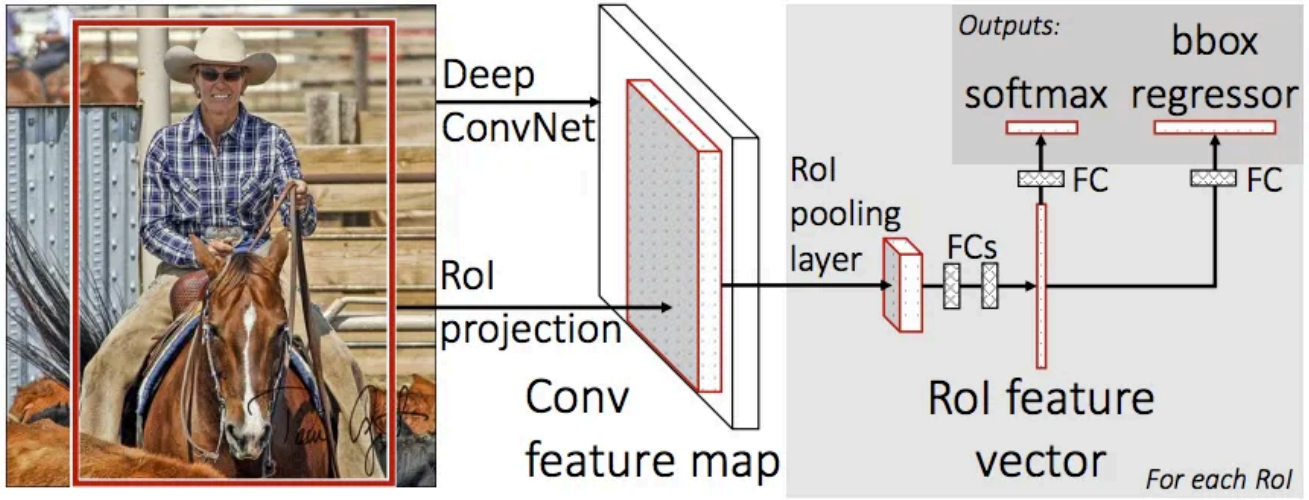
- Scan the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals
- Region candidates are warped to have a fixed size as required by CNN.
- Run a pre-trained CNN on top of each of these region proposals
- Take the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regression to tighten the bounding box of the object, if such an object exists.



R-CNN: Problems

- Generating the CNN feature vector for every image region (N images * 2000).
- It has to train three different models separately - the CNN to generate image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train.
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]

Fast R-CNN

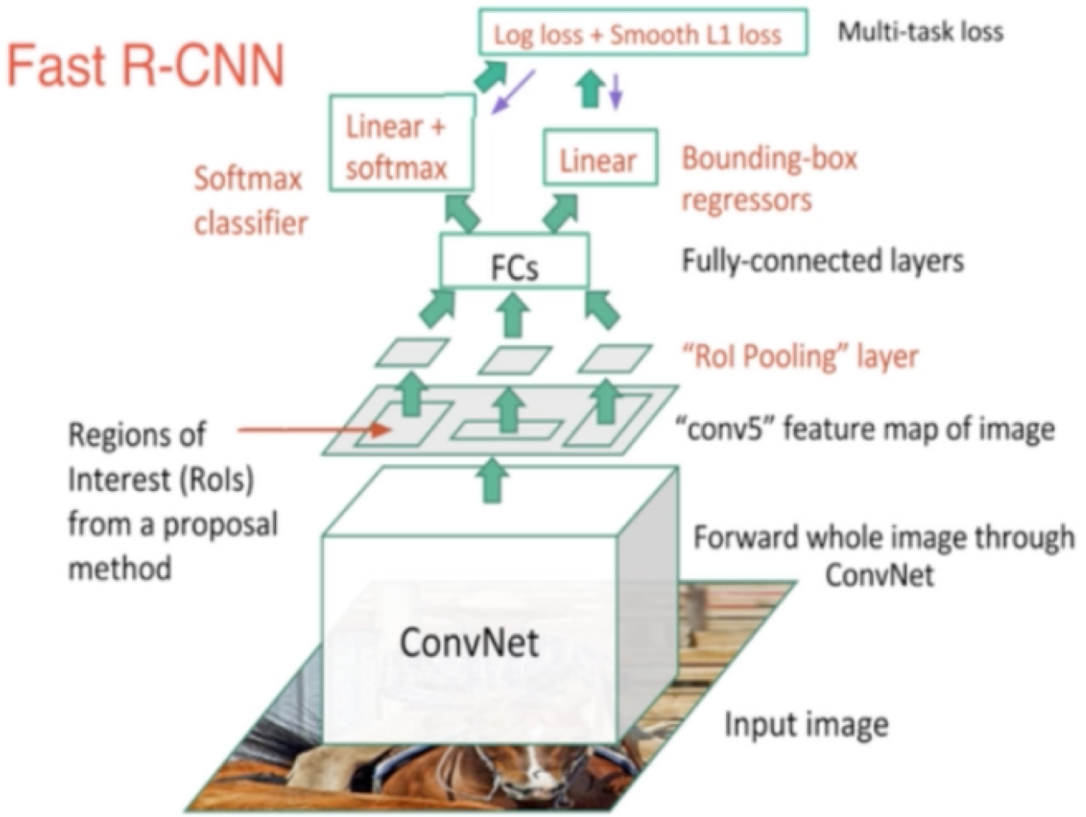


- Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
- From the convolutional feature map, we identify the region of proposals and warp them into squares
- By using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.
- From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

Fast R-CNN

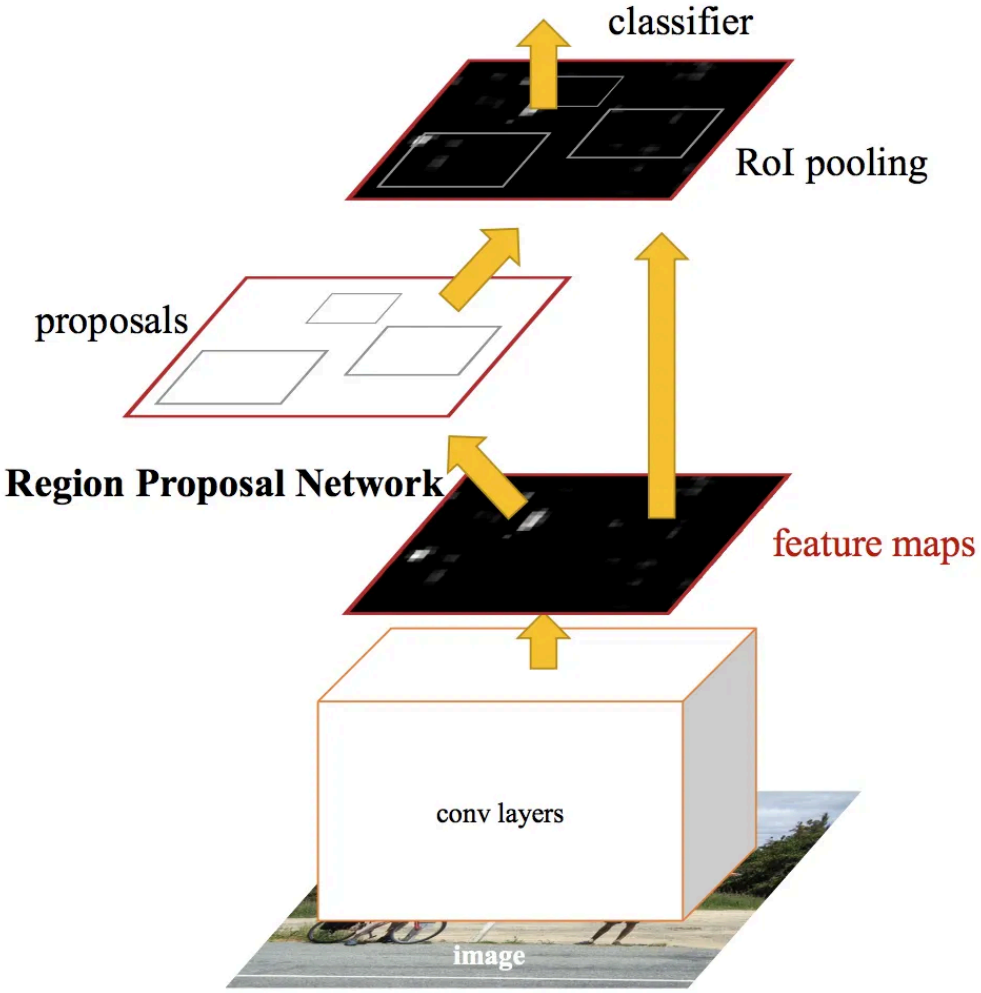
Fast R-CNN improved R-CNN on its detection speed through three main augmentations:

- Performing feature extraction over the image before proposing regions, thus onl running one pre-trained CNN over the entire image instead of 2000 CNN's over 2000 overlapping regions
- Replacing the SVM with a softmax layer, thus extending the neural network for predictions instead of creating a new model
- Fast R-CNN is to jointly train the CNN, classifier, and bounding box regressor in single model.



Faster R-CNN

- Both algorithms(R-CNN & Fast R-CNN) uses selective search to find out the region proposals.
- Selective search is a slow and time-consuming process affecting the performance of the network.
- Therefore, Shaoqing Ren et al. came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

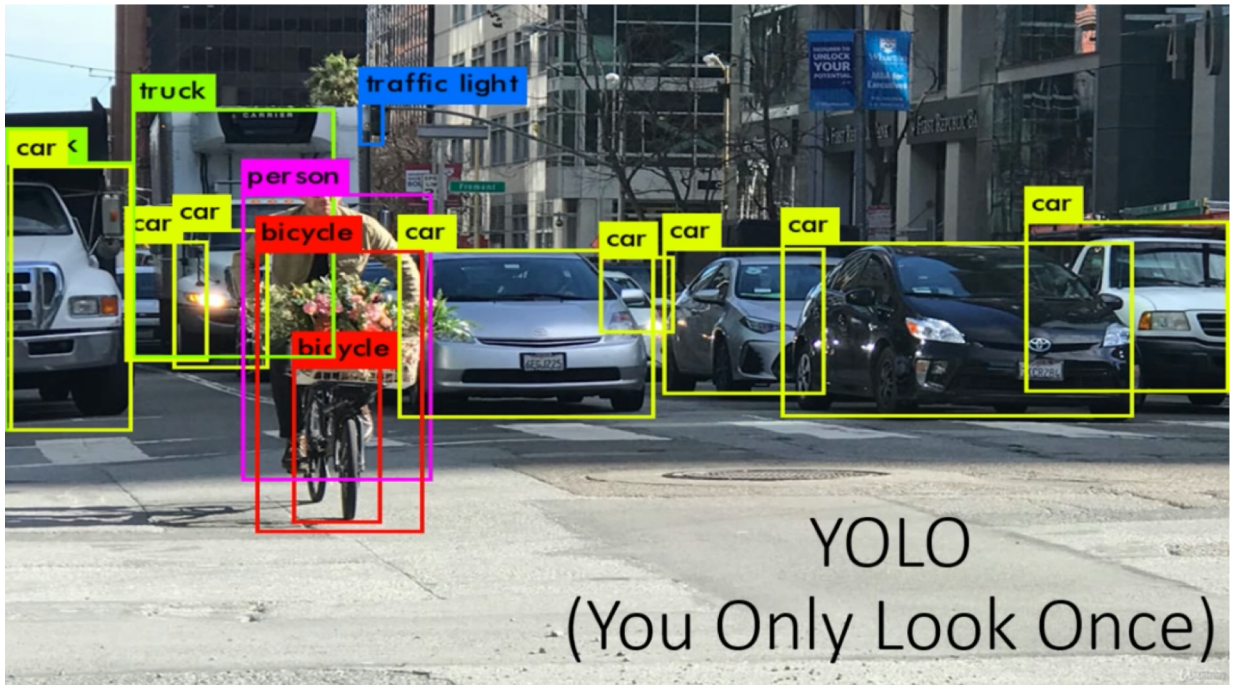


YOLO

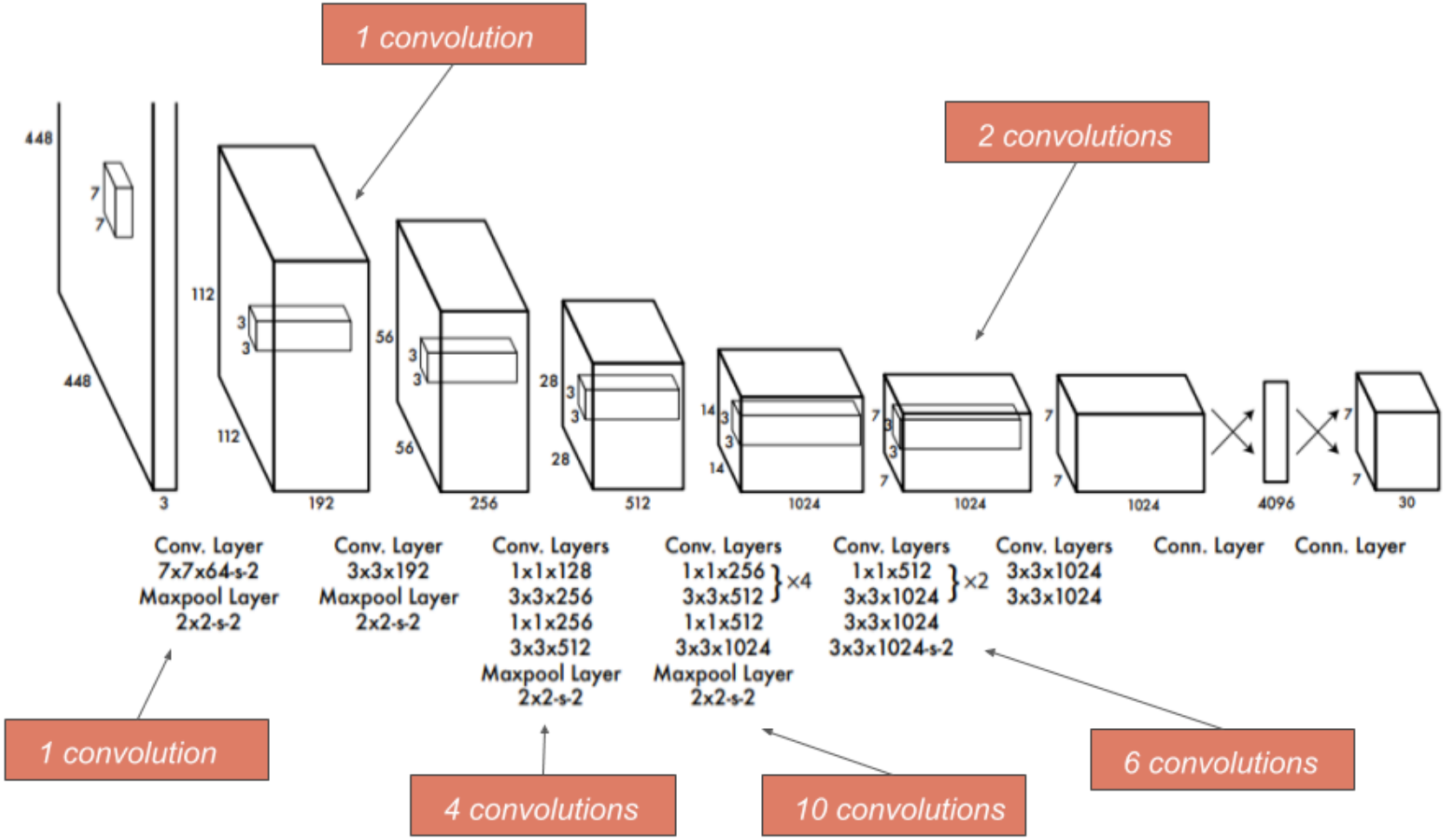
- All of the previous object detection algorithms use regions to localize the object within the image.
- The network does not look at the complete image.
- Instead, parts of the image which have high probabilities of containing the object.
- YOLO or You Only Look Once is an object detection algorithm much different from the region based algorithms seen above.
- In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

YOLO

The key idea behind YOLO is to perform object detection in real-time with a **single forward pass** through the neural network, making it very efficient and suitable for applications where speed is critical.

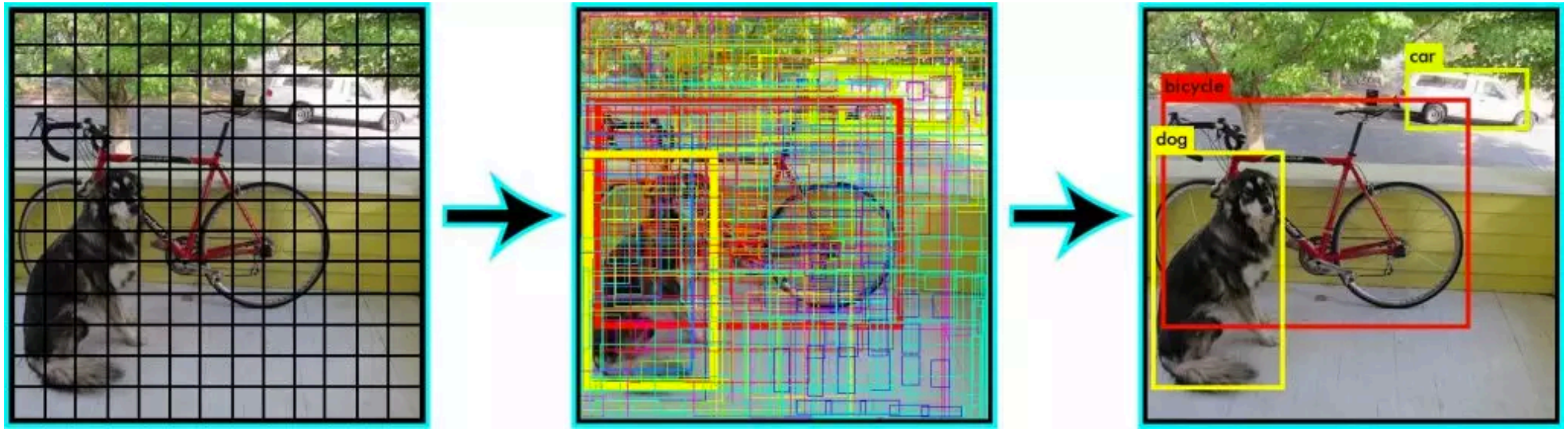


YOLO

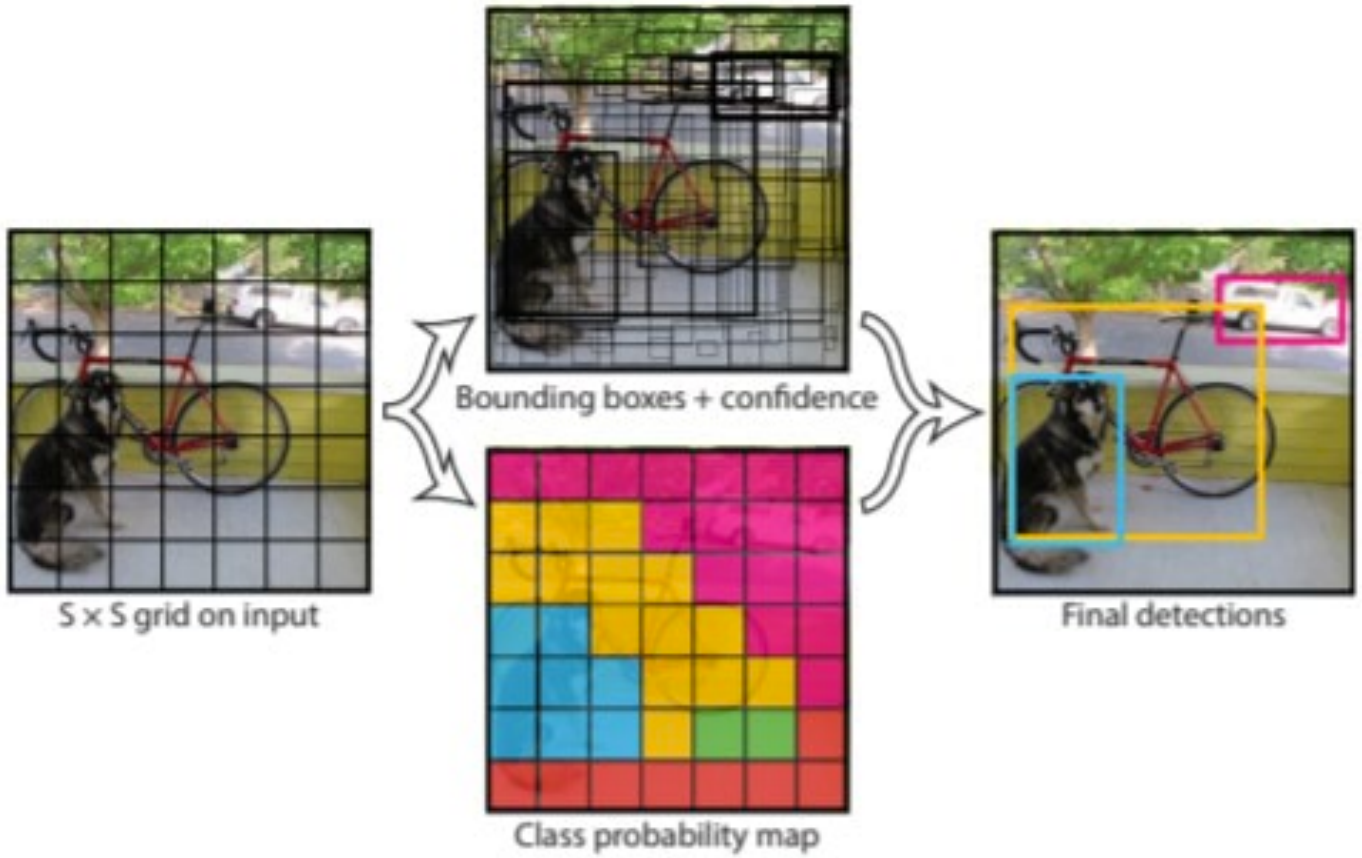

1 convolution
2 convolutions
448
448
7
7
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer
1 convolution
4 convolutions
10 convolutions
6 convolutions

YOLO

- How YOLO works is that we take an image and split it into an SxS grid, within each of the grid we take m bounding boxes.
- For each of the bounding box, the network outputs a class probability and offset values for the bounding box.
- The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

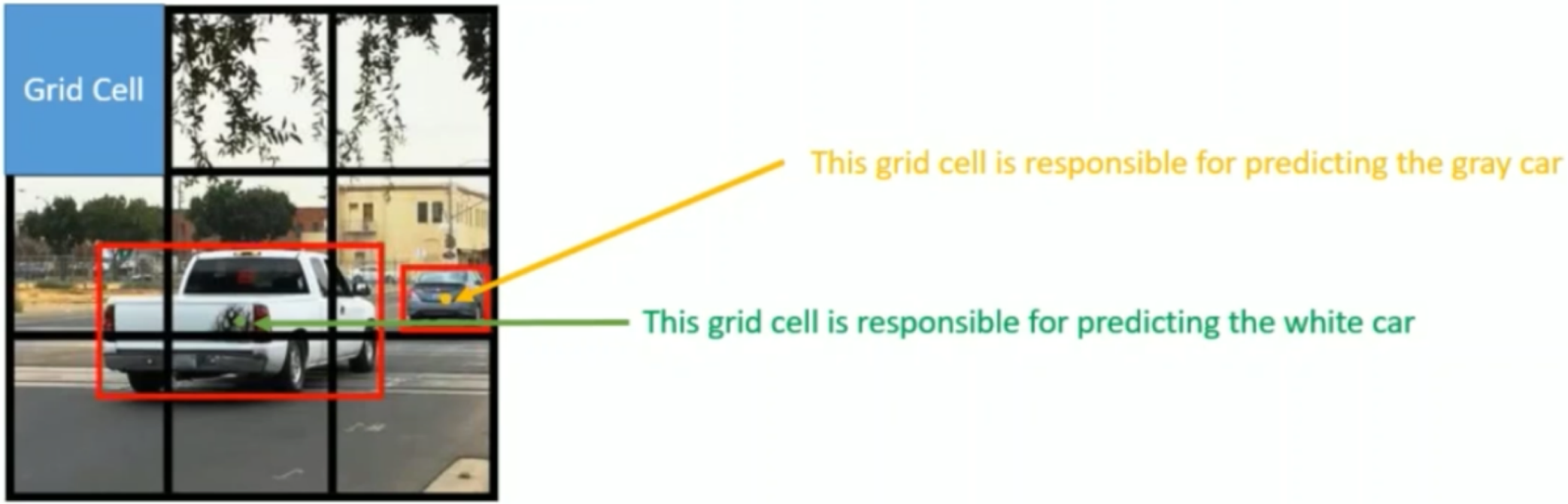


YOLO


Bounding boxes + confidence
S × S grid on input
Final detections
Class probability map

YOLO

- YOLO divides the input image into an S×S grid. Each grid cell predicts only **one** object.
- Let's look at this on a smaller scale. Divide this image into 3x3 Grid Cells (9 Grid Cells), and assign the center of the object to that grid cell. This grid cell is responsible for predicting the object.



YOLO

For each grid cell

- The labels we have for training are:

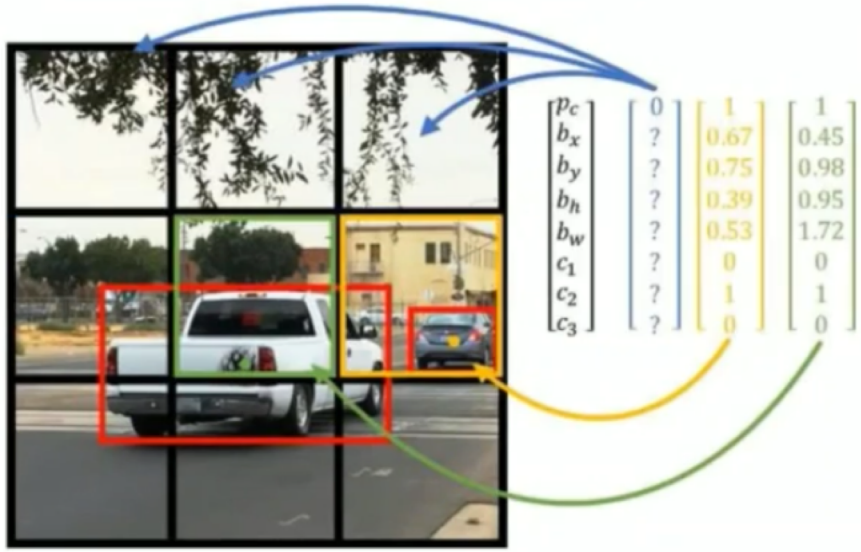
3 x 3 x 8

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

If the grid cell contains an object

The bounding boxes of the object (if there is)

The number of classes (assume 3 for this example)



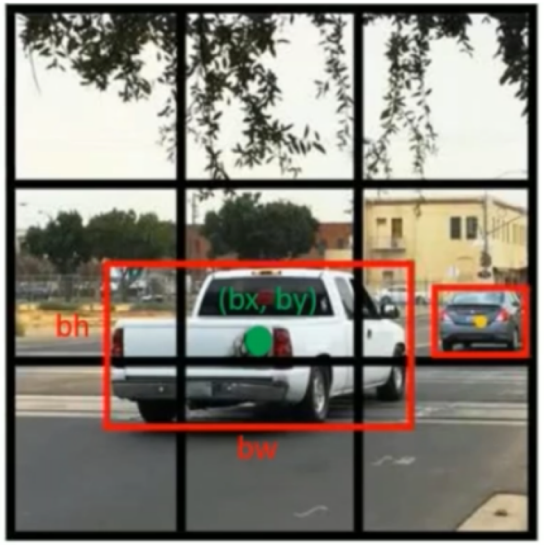
p_c	0	1	1	Is there an object?
b_x	?	0.67	0.45	Bounding box
b_y	?	0.75	0.98	
b_h	?	0.39	0.95	
b_w	?	0.53	1.72	
c_1	?	0	0	Class labels
c_2	?	1	1	
c_3	?	0	0	

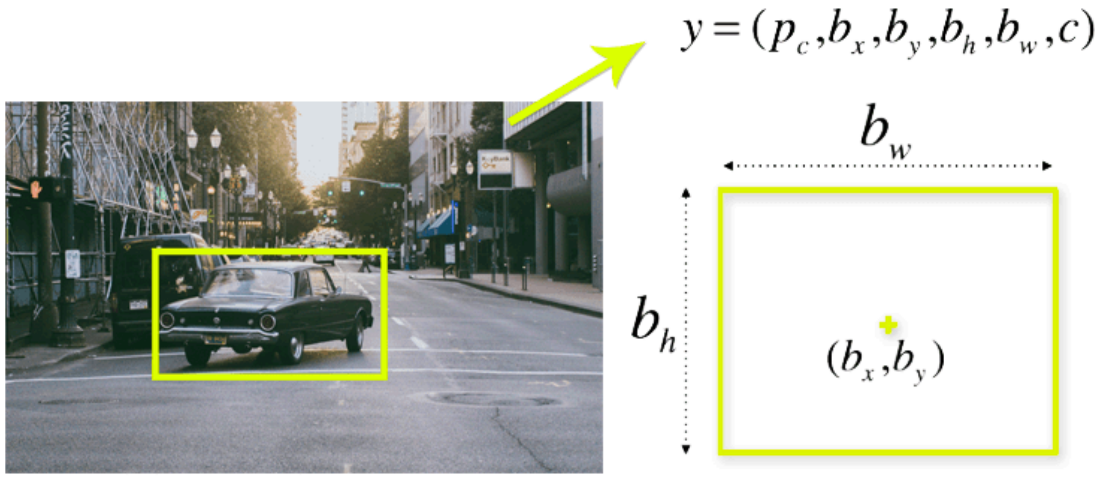
For those grid cells with no object detected, it's pc = 0 and we don't care about the rest of the other values. That's what the "?" means in the graph.

YOLO

Bounding Boxes

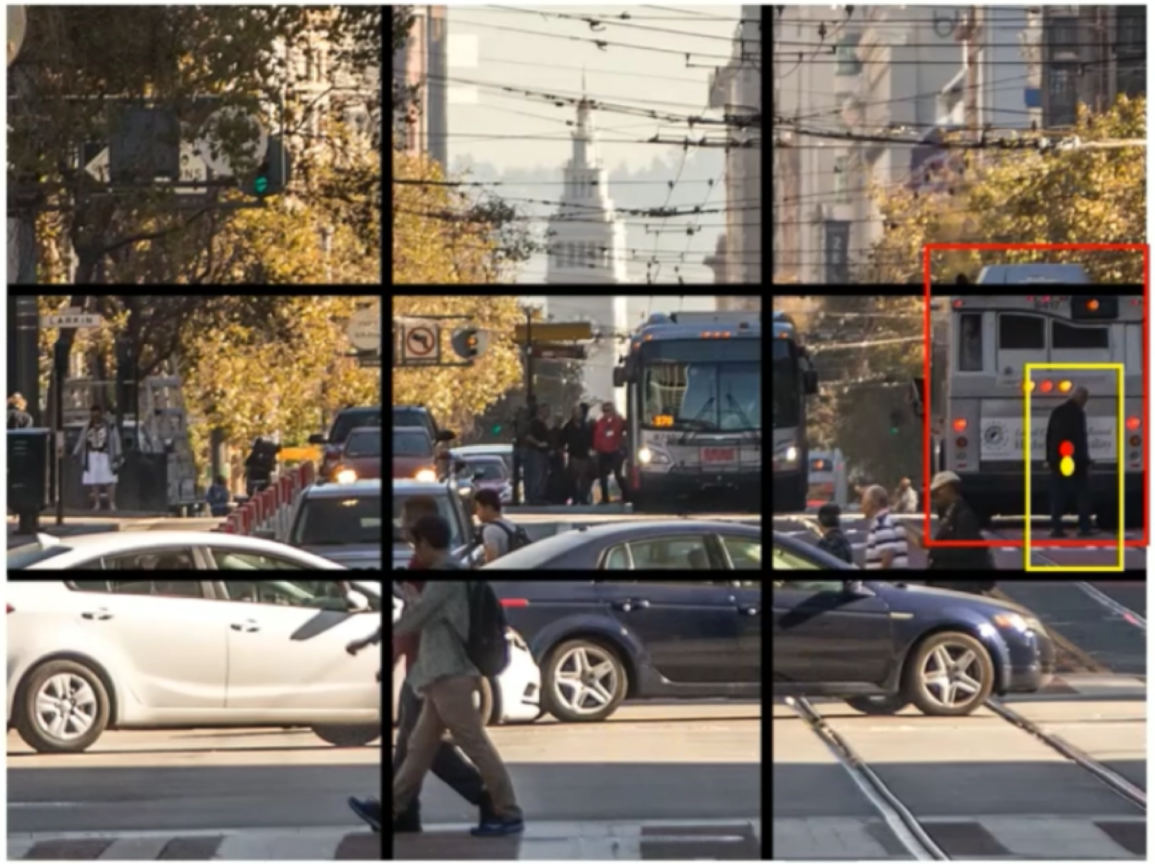
- bx: the center of the object according to the **x** coordinate, and has a value ranging from 0~1,
- by: the center of the object according to the x coordinate, and has a value ranging from 0~1
- bh: **height** of the bounding box, the value could be greater than 1,
- bw: **width** of the bounding box, the value could be greater than 1





YOLO

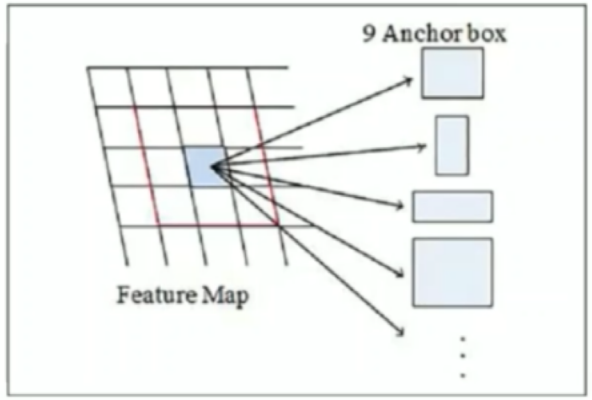
What if this happens?

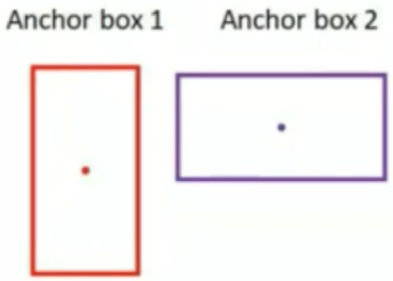


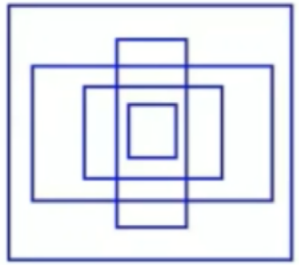
YOLO

Anchor Boxes

- There is a limitation with only having grid cells.
- Say we have multiple objects in the same grid cell. For instance, there's a person standing in front of a car and their bounding box centers are so close. Shall we choose the person or the car?
- To solve the problem, we'll introduce the concept of **anchor box**. Anchor box makes it possible for the YOLO algorithm to detect multiple objects centered in one grid cell.



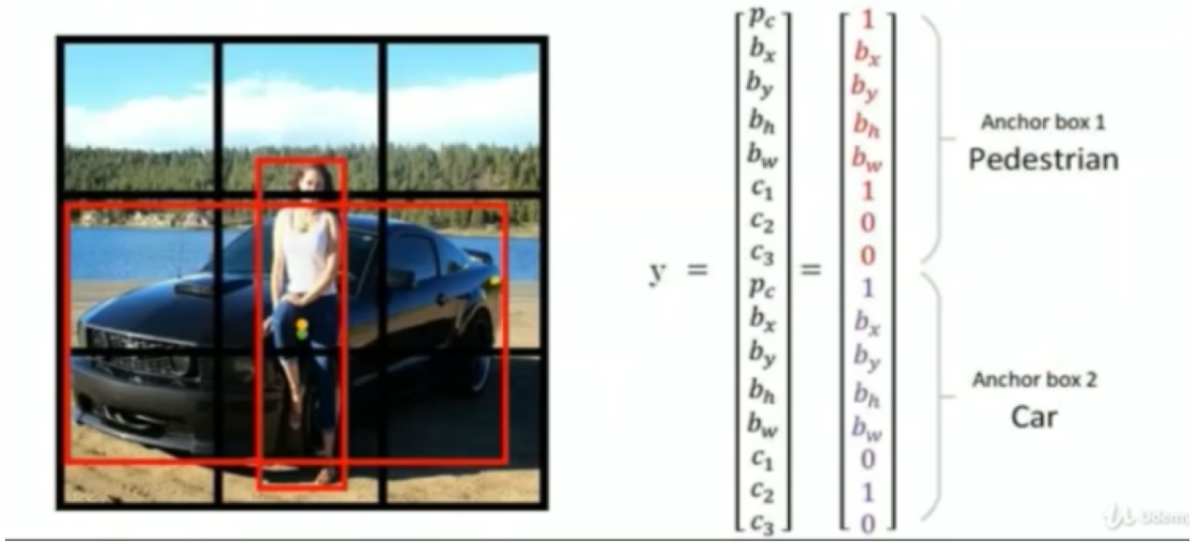




Notice that, in the image above, both the car and the pedestrian are centered in the middle grid cell.

YOLO

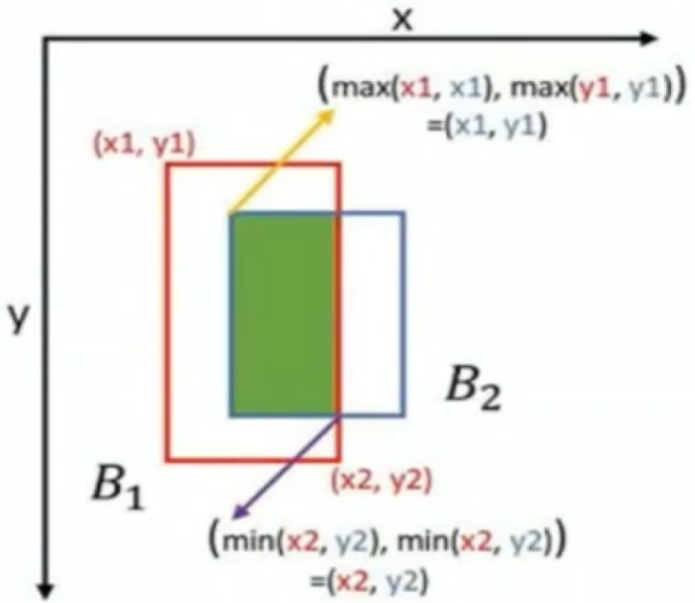
- For each grid cell, YOLO predicts multiple anchor boxes.
- Anchor boxes are predefined boxes with specific sizes and aspect ratios.
- These boxes represent potential shapes for objects in the image.
- YOLO predicts the coordinates of these anchor boxes' centers, their widths, heights, and the objectness score for each box.

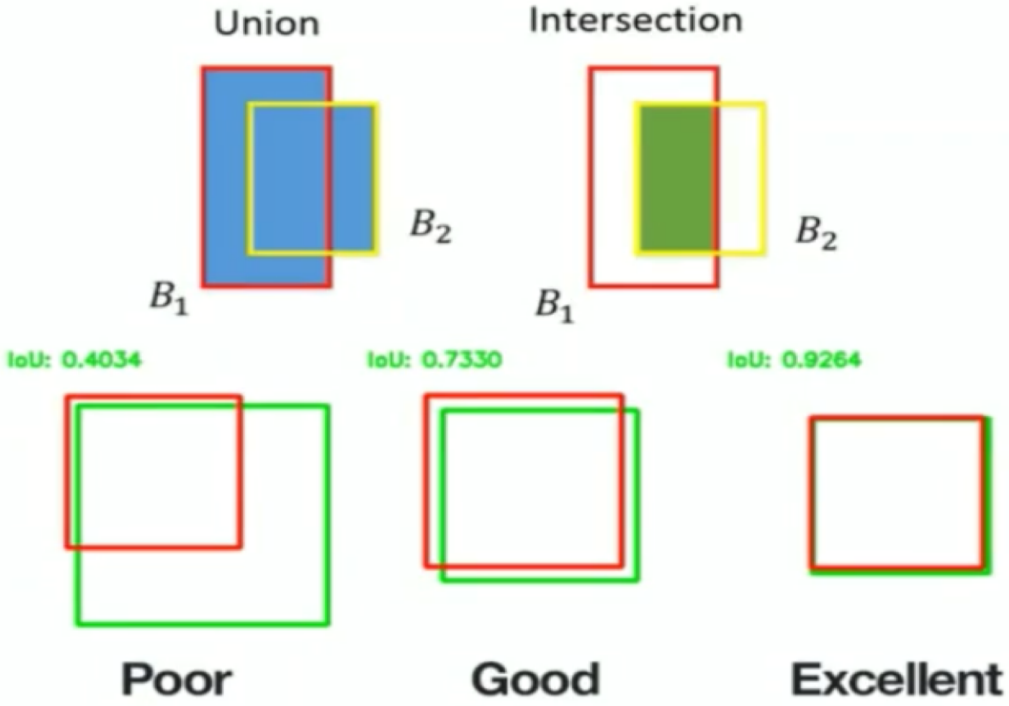


YOLO

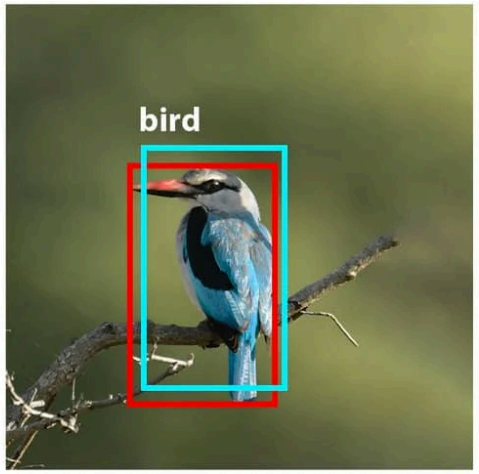
Evaluation metric

- Instead of defining a box by its center point, width and height, let's define it using its two corners (upper left and lower right): (x1, y1, x2, y2)
- To compute the intersection of two boxes, we start off by finding the intersection area's two corners.

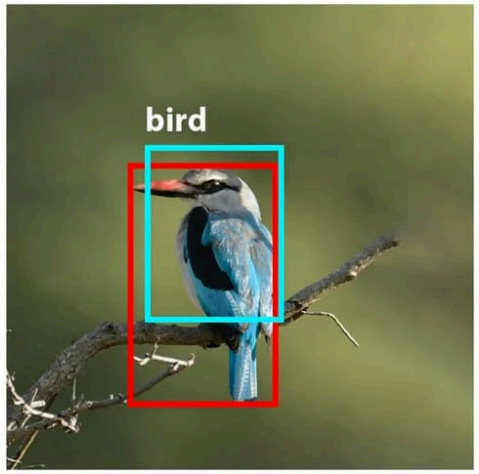




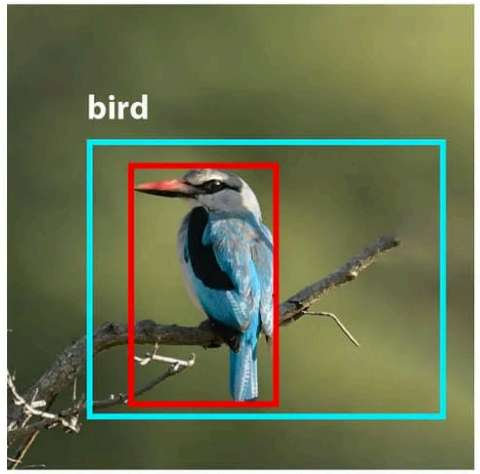
YOLO



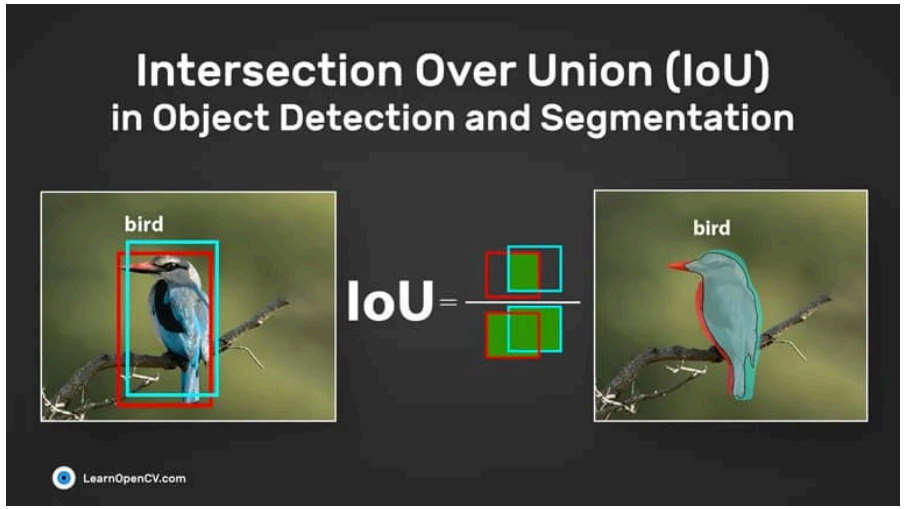
Model A

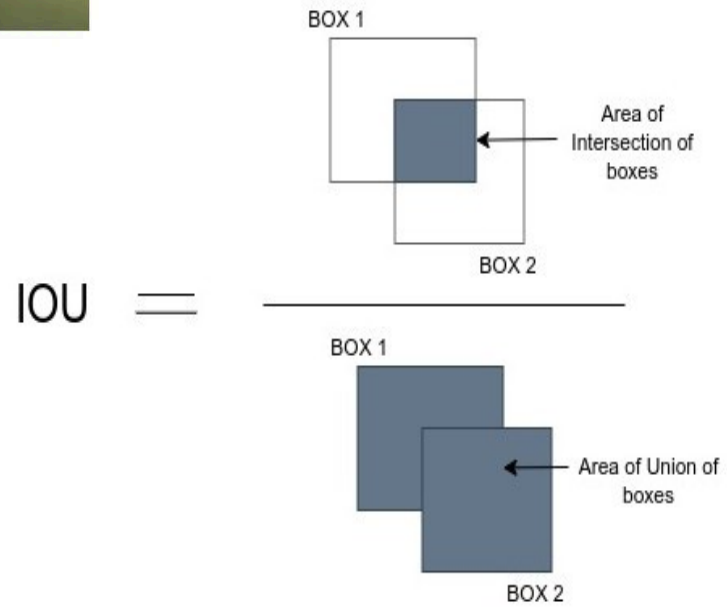


Model B

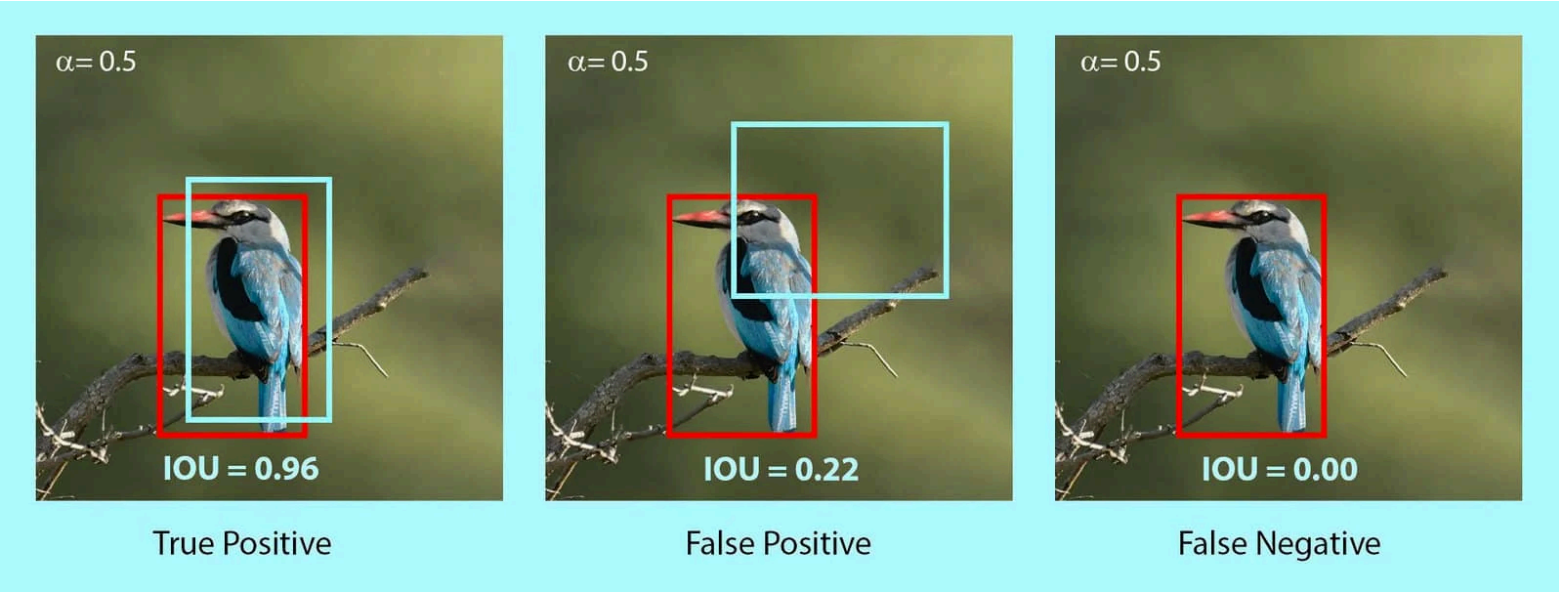


Model C





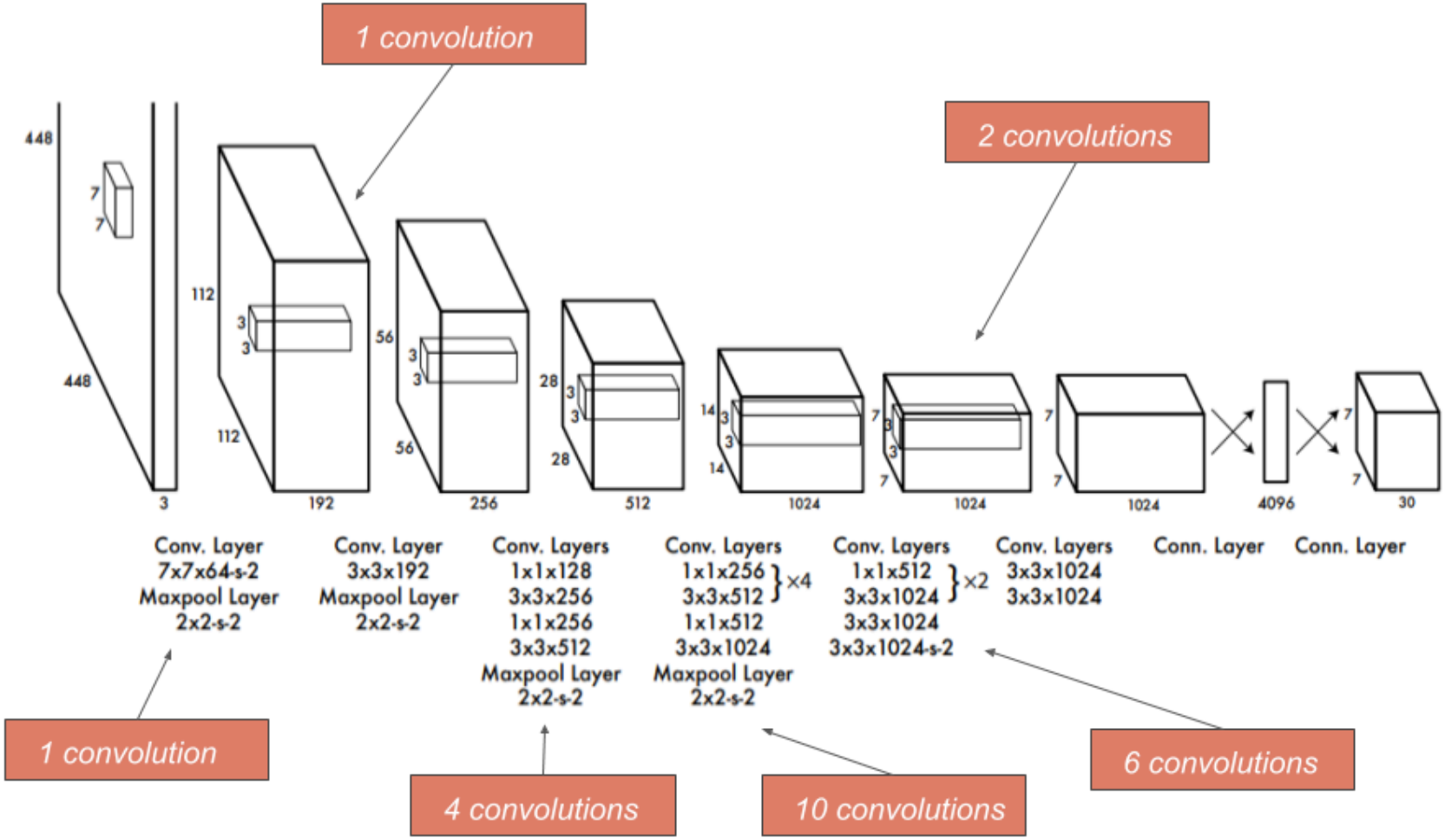
YOLO


α= 0.5
IOU = 0.96
True Positive
α= 0.5
IOU = 0.22
False Positive
α= 0.5
IOU = 0.00
False Negative

YOLO

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

YOLO


1 convolution
2 convolutions
448
448
7
7
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer
1 convolution
4 convolutions
10 convolutions
6 convolutions

YOLO

- The original YOLO (You Only Look Once) was written by Joseph Redmon in a custom framework called Darknet.
- **Darknet** is a very flexible research framework written in low level languages and has produced a series of the best realtime object detectors in computer vision: YOLO, YOLOv2, YOLOv3, and now, YOLOv4
- The original YOLO model was the first object detection network to combine the problem of drawing bounding boxes and identifying class labels in one **end-to-end** differentiable network.

R-CNN Vs YOLO

Computational Efficiency:

R-CNN Family: The R-CNN family involves **two-stage processing**. It first generates region proposals using techniques like selective search, and then it classifies and refines these proposals. This two-stage process can be computationally expensive, making real-time processing challenging.

YOLO: YOLO is designed as a **one-stage object detection model**. It predicts bounding boxes and class labels directly in a single pass through the neural network. This streamlined approach is computationally more efficient and well-suited for real-time applications.

R-CNN Vs YOLO

Speed:

R-CNN Family: The two-stage nature of R-CNN models, involving both region proposal and classification, can result in **slower inference times**.

YOLO: YOLO is known for its speed and efficiency. It can process images in real-time or near real-time, making it suitable for applications that require quick responses, such as autonomous vehicles or video surveillance.

Object Detection in Real-Time:

R-CNN Family: Due to its computational complexity, the R-CNN family can be challenging to deploy in real-time systems.

YOLO: YOLO is designed for real-time or near real-time object detection, making it a more practical choice for applications that require fast object recognition and tracking.