Object localization and detection

- **Image classification** — A single class is assigned to an image, commonly to the main object that is portrayed in the image. If an image contains a cat, the image will be classified as a 'cat'. With image classification, we do not know the precise location of the cat in the image nor can identify its limits on the visual compared to object localization, detection, or segmentation.

- **Object detection** — The objects within an image or a video are detected, marked with a bounding box, and then labeled. With object detection, the key signifier is the bounding box that draws a square or rectangle around the limits of each object.

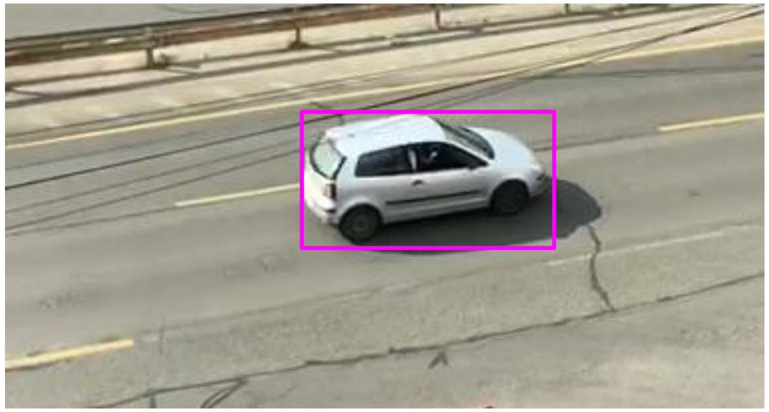
- **Localization** — With image/object localization, we are able to identify the location of the main subject of an image. However, image localization does not typically assign classes to the localized object as it considers the main subject instead of all of the present objects in a given frame.

Object localization and object detection

Classification (the entire image)

“car”

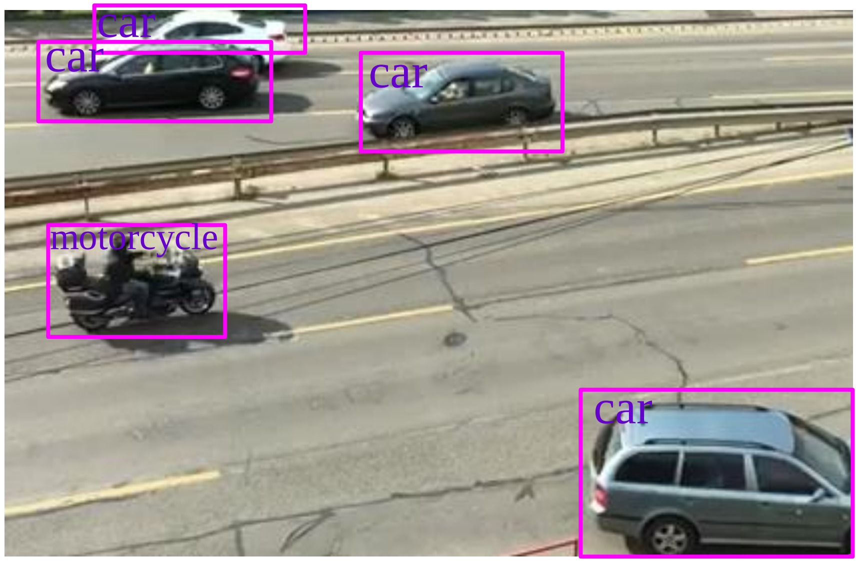
Localization + Classification



Where in the picture is the “car” ?

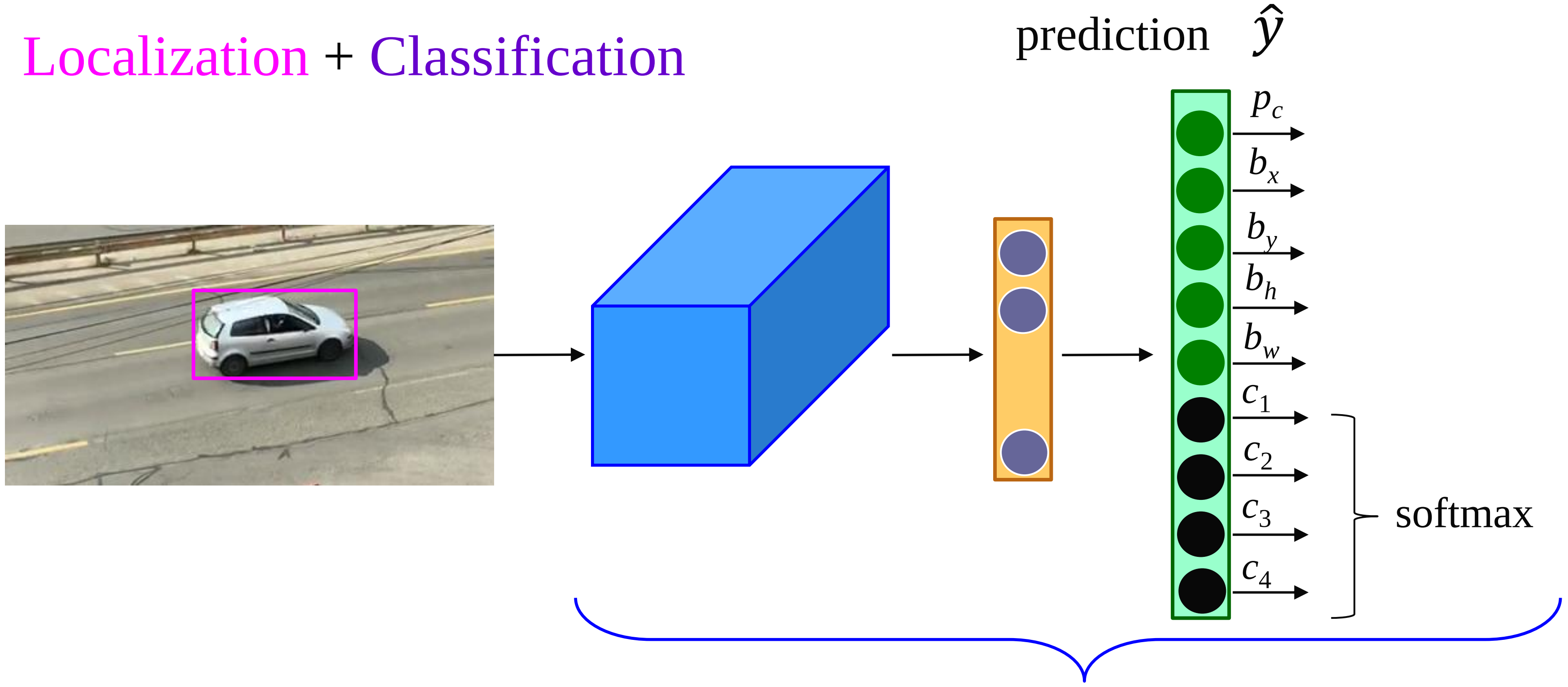
A **bounding box BB** is used to **localize** an object

Detection + Localization + Classification



Multiple objects in multiple categories in the same picture!

Localization + Classification



Consider 4 classes (c_1, …, c_4):
- car
- pedestrian
- bicycle
- motorcycle

$$\hat{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

p_c: probability to have on object (confidence)

b_x, b_y, b_h, b_w: bounding box coordinates

c_1, c_2, c_3, c_4: probability for each class

Localization + Classification

Example

Consider 4 classes:

- car
- pedestrian
- bicycle
- motorcycle

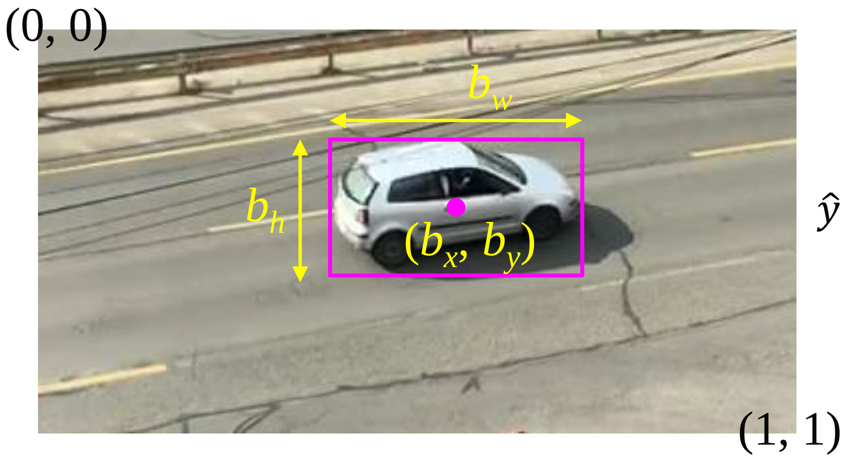
$$\hat{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

probability to have on object

bounding box coordinates

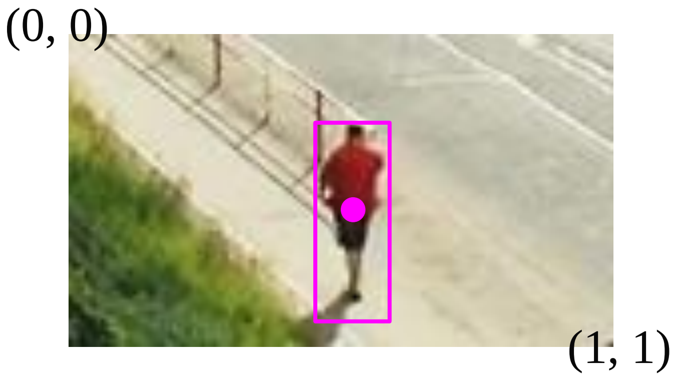
probability for each class

Normalized image size: [0, 1]



$$\hat{y} = \begin{bmatrix} 1 \\ 0.6 \\ 0.45 \\ 0.35 \\ 0.32 \\ 0.87 \\ 0.02 \\ 0.05 \\ 0.06 \end{bmatrix} \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

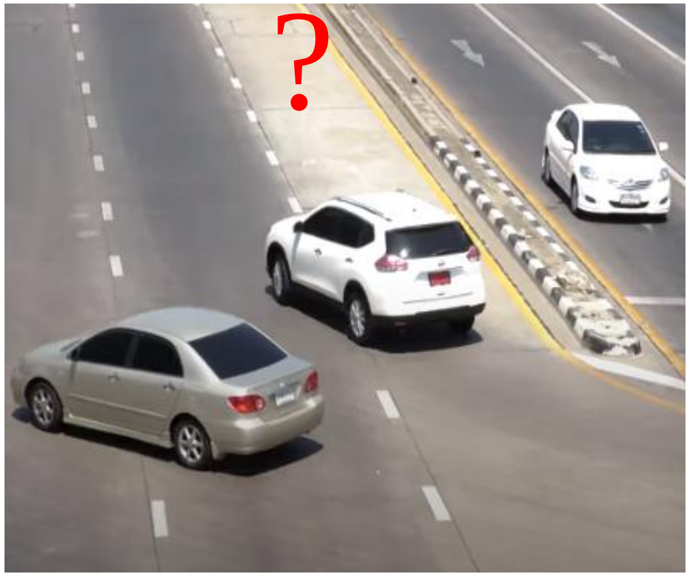


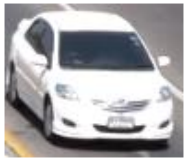
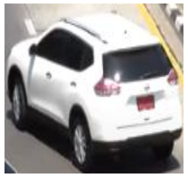
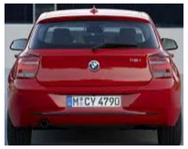
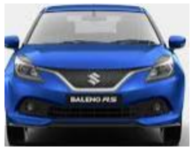
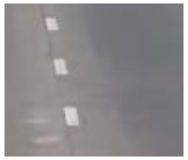
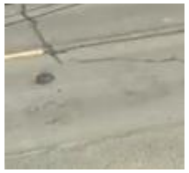
$$\hat{y} = \begin{bmatrix} 1 \\ 0.52 \\ 0.53 \\ 0.61 \\ 0.14 \\ 0.05 \\ 0.70 \\ 0.15 \\ 0.1 \end{bmatrix} \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

Object detection

Create a labeled training set (positive and negative): the object (car) is (pretty much) the entire image

Car detection

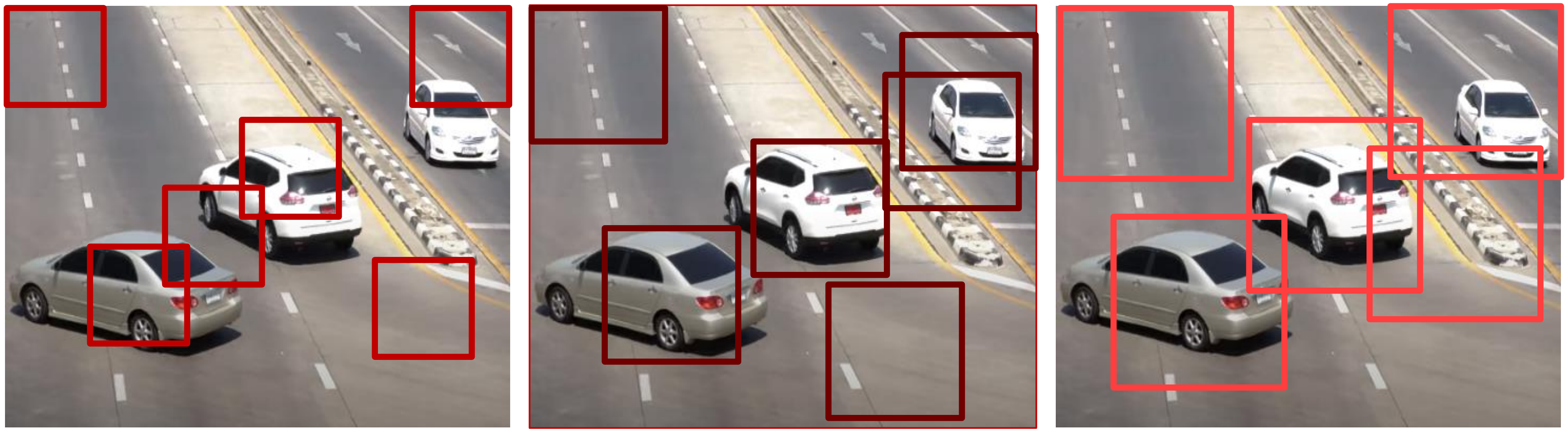


in x	**out** y
	1
	1
	1
	1
	0
	0

Train a ConvNet with such a training set to recognize if there is a car in each input image

Object detection: Sliding window Car detection

1. Use a window to search the input image for the objects of interest
2. Input the content of each window to the input of ConvNet, as an image, to see if there is/isn't a car
3. Slide windows using a certain stride, to sequentially search the entire original input
4. Repeat with a larger window



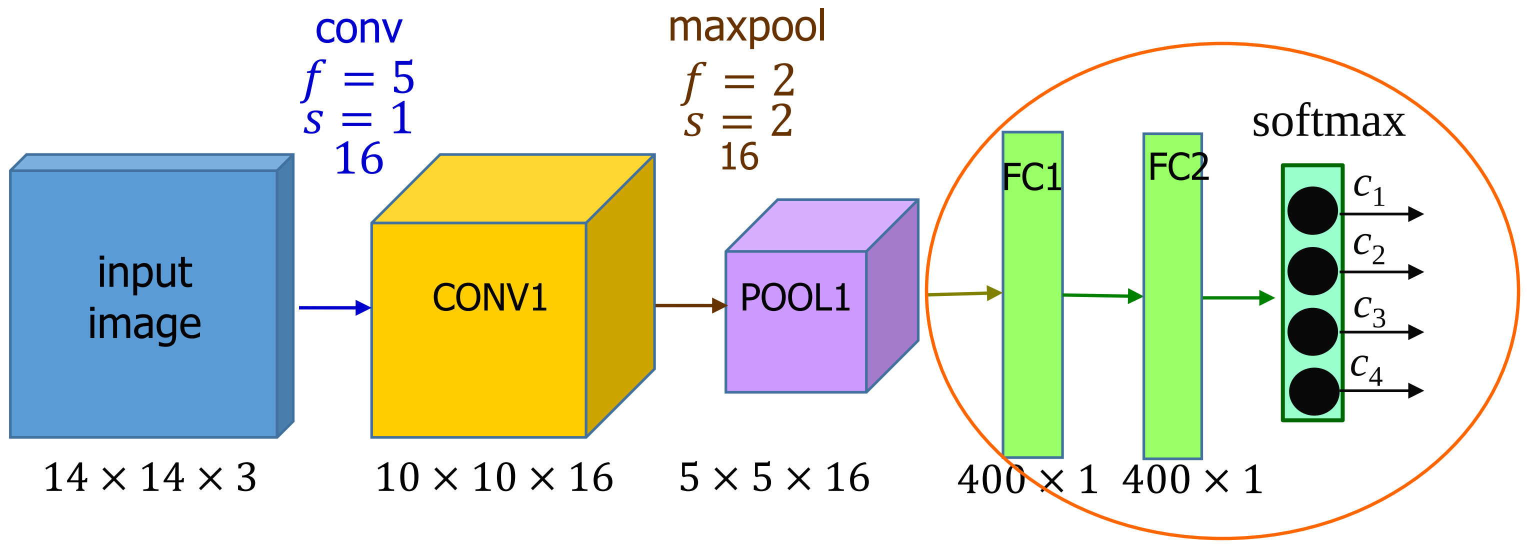
The ConvNet should be used sequentially, for the content of each window, for all window sizes

Huge disadvantage: very high computation cost for high accuracy: multiple window size, small stride - **infeasible slow**

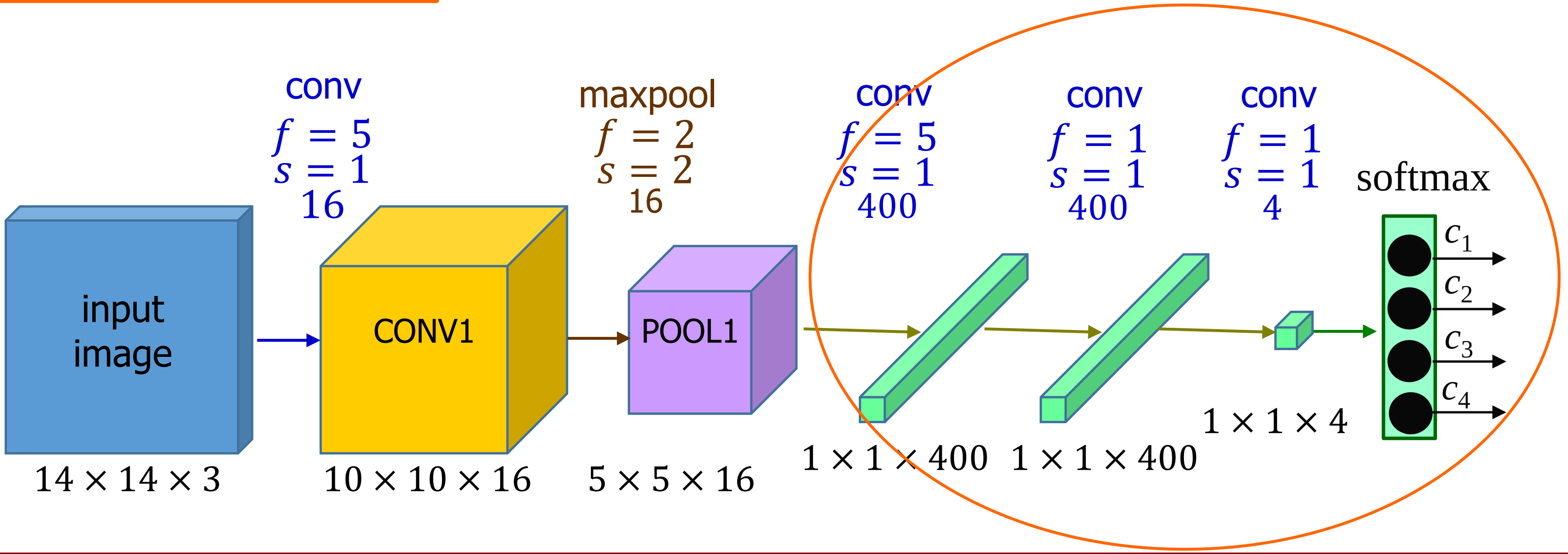
Using a large stride, the processing speed increases, but the coarser granularity may hurt the performance (can miss some objects)

Convolutional implementation of Sliding Window

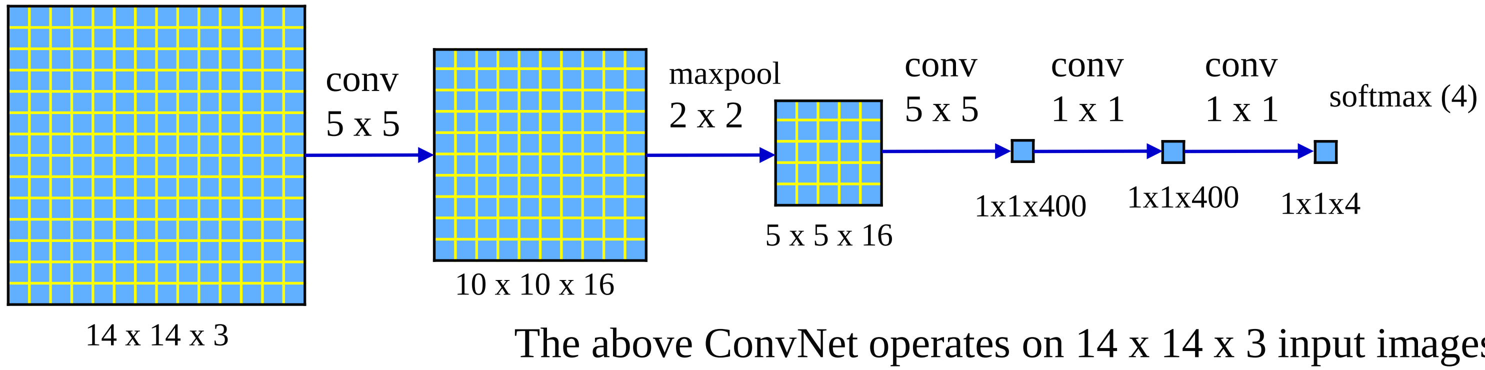
Assume a
14 x 14 sliding
window;
4 classes



FC layers → CONV layer



Convolutional implementation of Sliding Window



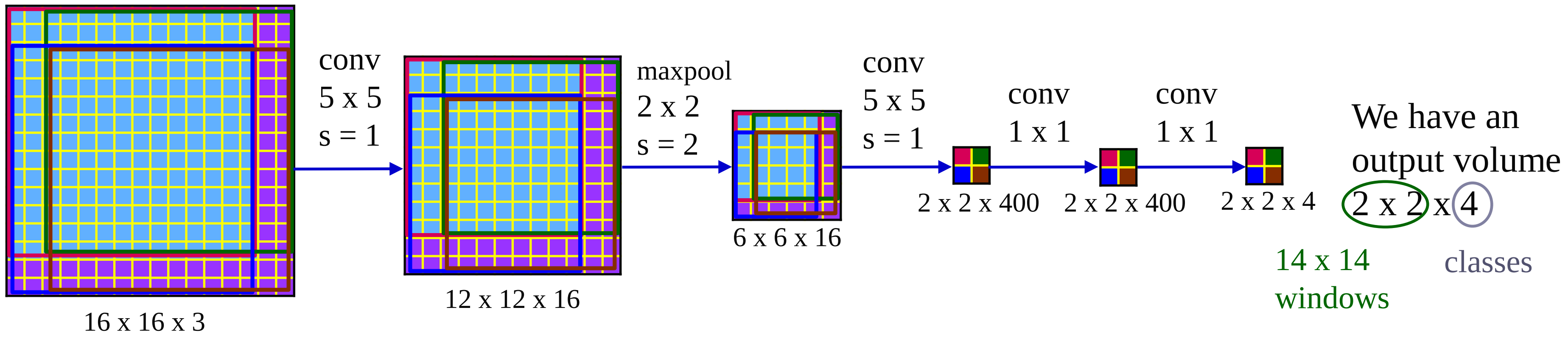
Use here only the front face of the volumes (for graphical representation)

The above ConvNet operates on 14 x 14 x 3 input images;
Now consider a 14 x 14 window as a sliding window.

Let's use a sliding window (14 x 14, stride = 2), on a 16 x 16 x 3 input images

- treat it **sequentially** using **4** (14 x14) windows: red, green, blue, brown.

What about using a ConvNet **only once** on the entire 16 x16 x 3 input image?



Make **all the predictions**, for all windows, at the same time, in **one forward propagation** and it may recognize the position (location – initial window) of the objects – objects detection.

Convolutional implementation of Sliding Window

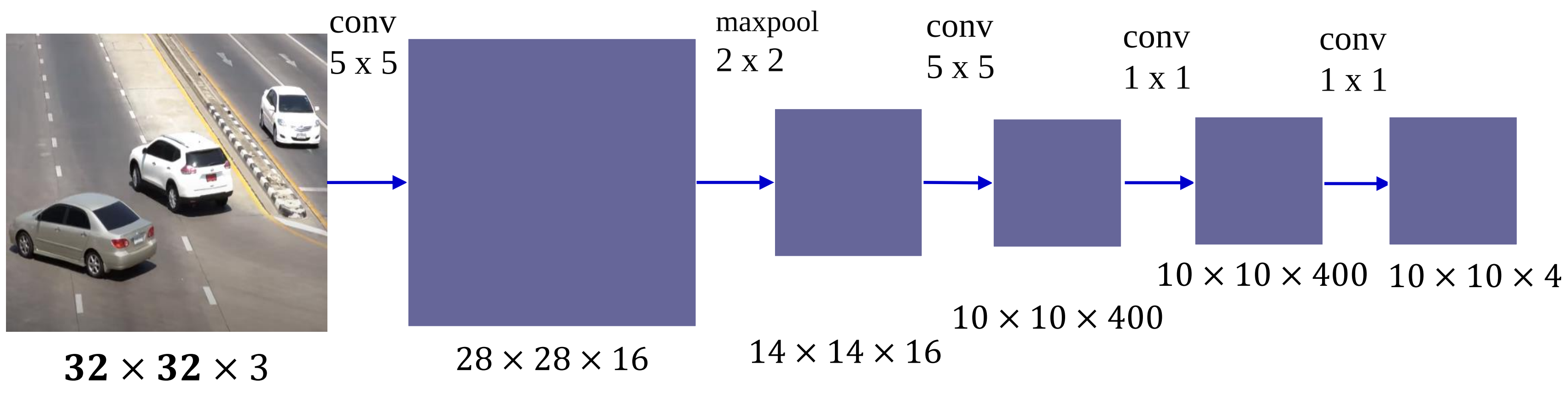
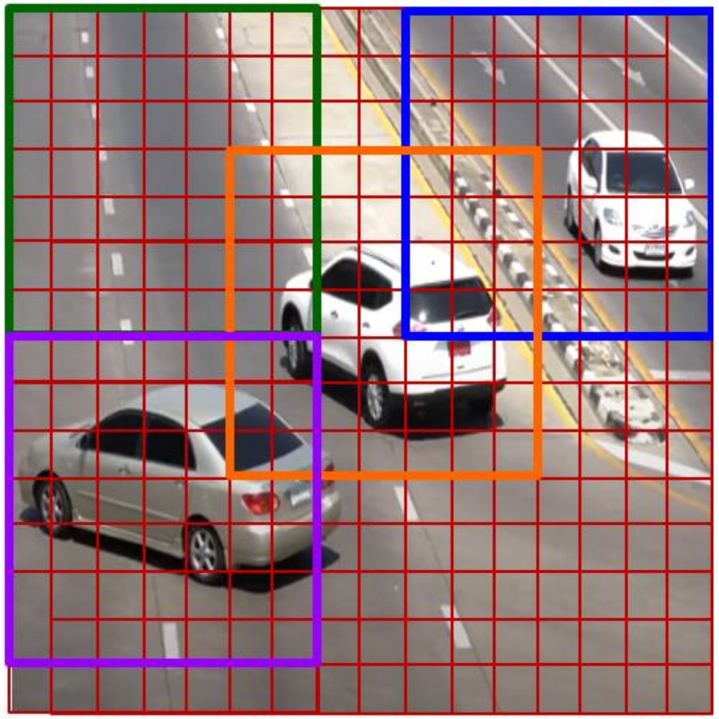


image: 32 x 32 x 3
window size: 14 x 14
stride = 2
# windows: 10 x 10

one forward pass

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor = \left\lfloor \frac{32-14}{2} + 1 \right\rfloor$$

10×10



We have an output volume:
10 x 10 x 4

Weakness: the window size and position that detects a car (object) is not an accurate prediction of the bounding box.

Bounding box prediction: YOLO

YOLO - You Only Look Once (one forward pass)

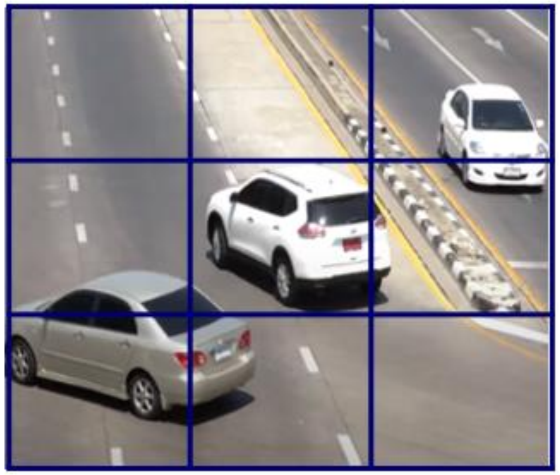
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, You Only Look Once: Unified, Real-Time Object Detection, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

1. Use a grid to split the original image in smaller cells (images).

   For simplicity consider here a 3 x 3 grid, non-overlapping windows. In a practical implementation a finer grid should be used, for example a 19 x 19 grid.

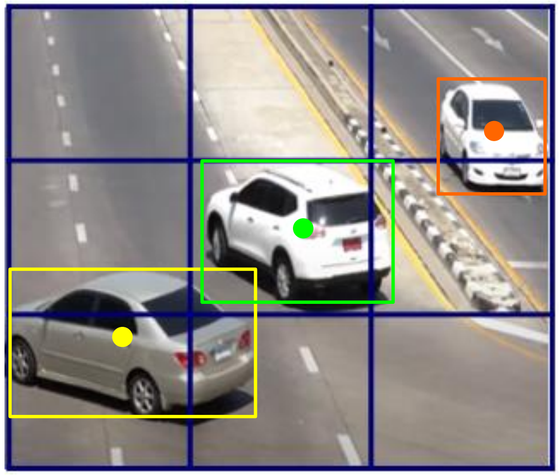
2. Run a classification and localization algorithm for **each grid cell of the initial image**.

- If 4 classes are considered, use a **training label (*y*)** for each grid cell.
  - We'll have 9 labels with the size 9.
- For each object, YOLO considers the object center, and the cell containing the center is responsible for that object detection, even if the object spans multiple grid cells.

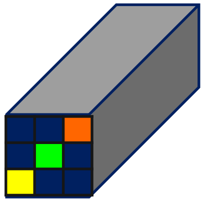


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

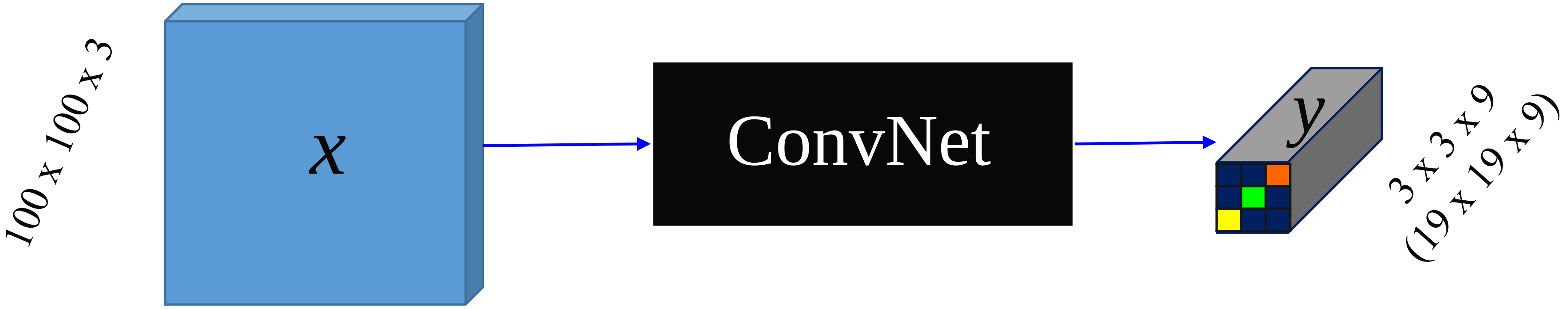
Bounding box prediction: YOLO



The target (output) is going to be: 3 x 3 x 9



$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$



Use backpropagation to train the ConvNet to map from any input image ***x*** to the output volume ***y***. The ConvNet will output accurate bounding boxes.

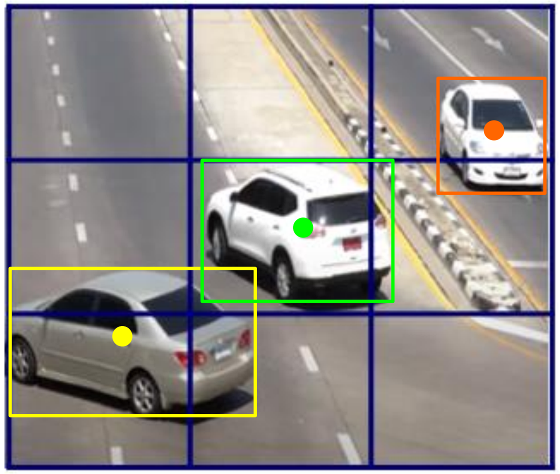
For prediction, the ConvNet runs only once for the entire image, due to its convolutional implementation. The algorithm is fast, so it can be used for **real-time applications**.

Multiple objects in a grid cell?
Usually, a finer grid cell will be used (19 x 19) – is this enough?

Bounding box prediction: YOLO

How can the bounding box be coded?

One solution is to use a reference to the top left corner (0, 0), and the bottom right corner (1, 1) inside of each grid cell.



$$y = \begin{bmatrix} 1 \\ 0.75 \\ 0.8 \\ 0.8 \\ 0.6 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0.6 \\ 0.5 \\ 0.9 \\ 1.1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0.75 \\ 0.2 \\ 0.97 \\ 1.4 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

$b_x, b_y \in (0, 1)$

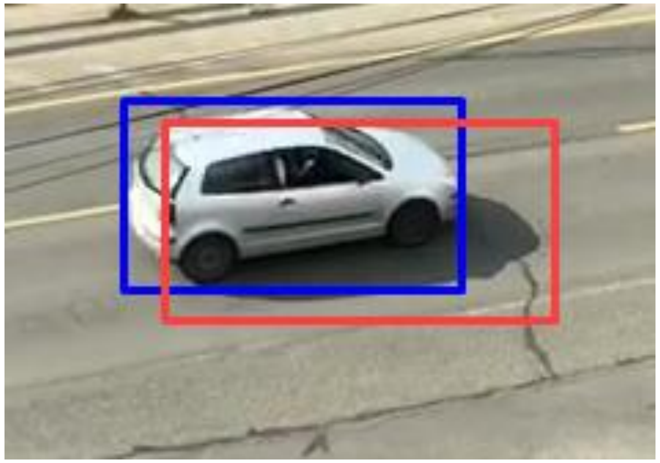
$b_h, b_w > 0$, and they can be > 1 if the bounding box has one dimension greater that the dimension of a grid cell

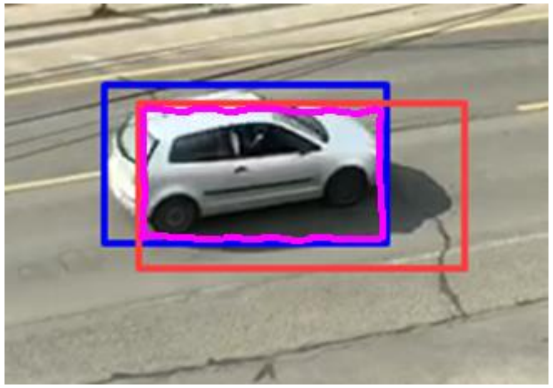
IoU – Intersection over Union

- ❖ Evaluate the object detection algorithm
- ❖ Used with **non-max suppression**

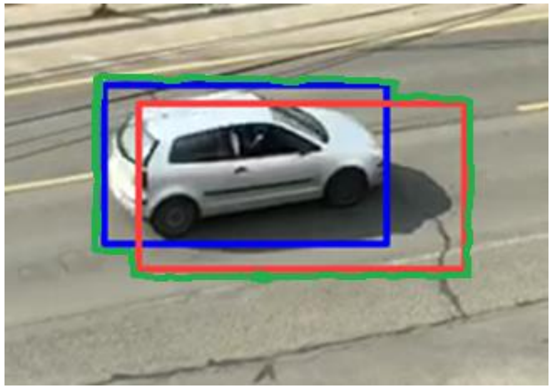
Computes the intersection area over the union area of two bounding boxes (bb):

- Detected bb
- Ground truth bb





$$\text{IoU} = \frac{\text{size of } \square}{\text{size of } \square}$$



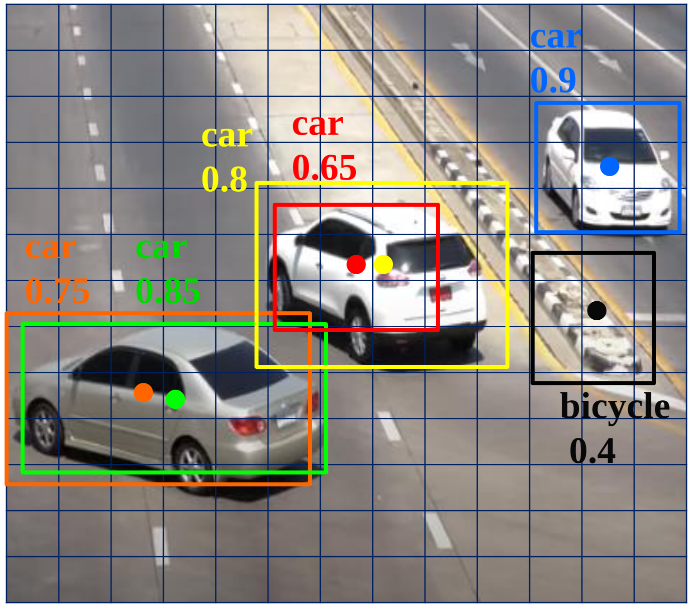
Usually, the object localization is **correct** if **IoU ≥ 0.5**

IoU

- a measure of the overlap between two bounding boxes
- a way of measuring how similar two boxes are to each other

Non-max suppression

13 x 13 grid cell



- One common problem in object detection is multiple detection of the same object

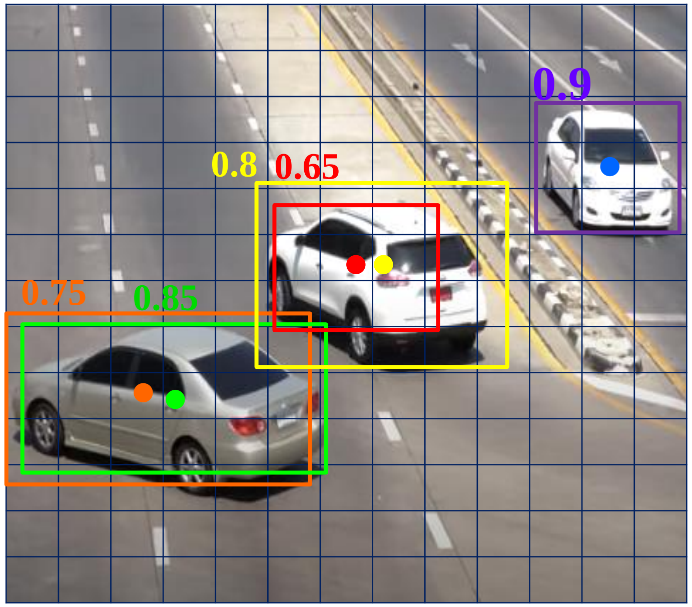
More than one cell "considers" that it contains the center of an object (with different bb), so those cells become responsible for the object detection, hence multiple detection of the same object.

The algorithm output 6 predictions at first, even if there are only 3 objects in the input image

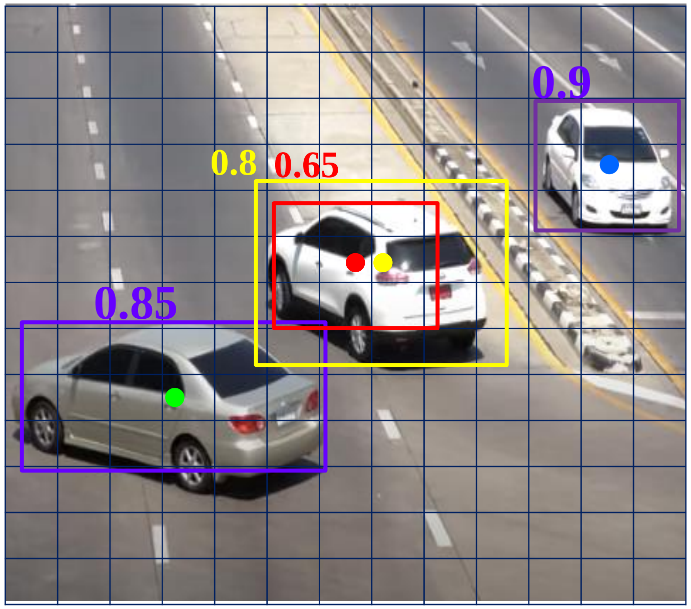
Now, it is necessary to "**clean up**" for multiple detections these 6 predictions

0. Discard all predictions with a low p_c and keep the ones with a higher p_c
   - in our case consider the **threshold for p_c** to be 0.6; The black bb is discarded.
1. Next, consider the detection with the largest value of probability of detection (confidence) – the most confident detection,
   - in our case: blue one: 0.9 – we have the 1st detected car – keep it
2. Non-max suppression looks at all the remaining bb and computes **IoU for all, against** the last detected object . The ones with a high overlap (high IoU) with the current one will get suppressed.
   - no overlap at all (IoU = 0) for blue with all remaining bbs. Nothing is suppressed.

Non-max suppression

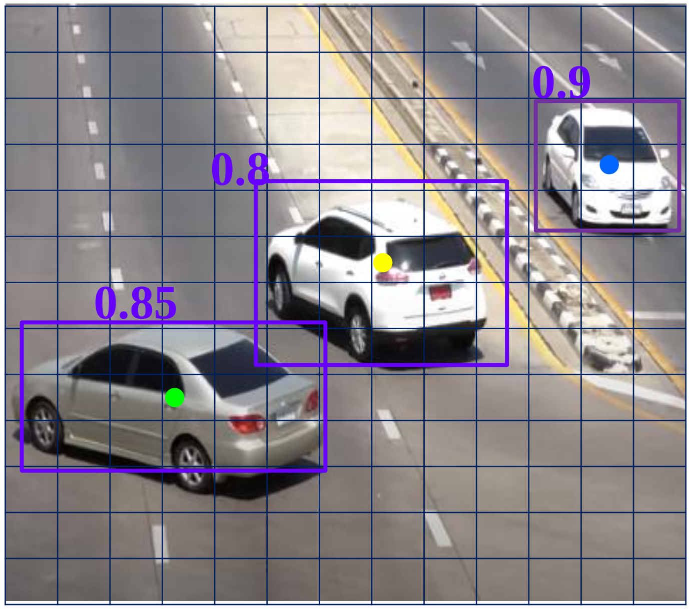


3. From the remaining detections, consider the one with the largest p_c and keep it as current bb
   - green: 0.85 – we have a 2nd detected car
4. Non-max suppression looks at all the remaining bb, and the ones with a high overlap (high IoU) with the current one will get suppressed
   - high overlap only with orange (IoU = 0.92); the orange one is suppressed (non-max: 0.75<0.85)



5. From the remaining detections consider the one with the largest p_c and keep it as current bb
   - yellow: 0.8 – we have a 3rd detected car
6. Non-max suppression looks at all the remaining bb, and the ones with a high overlap (high IoU) with the current one will get suppress
   - high overlap only with red (IoU = 0.7); the yellow one is suppressed (non-max: 0.65<0.8)

Non-max suppression



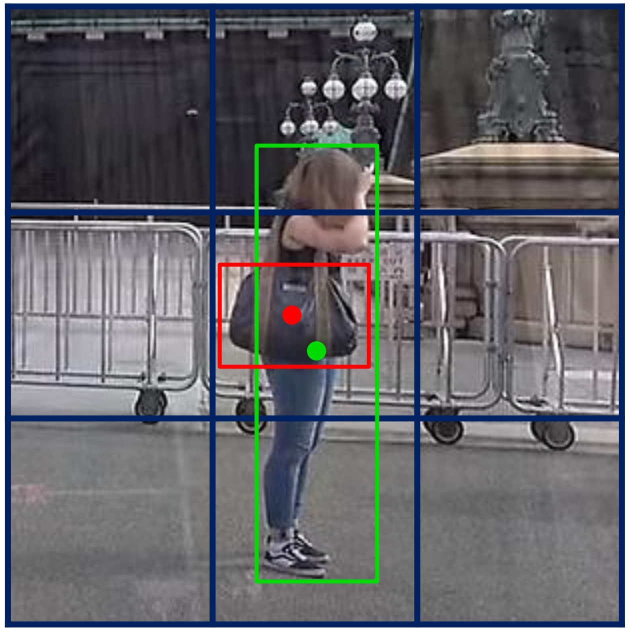
- This is the final detections with unique detection for all three cars, and their bbs
- **Non-max suppression:** output the maximum largest probabilities classifications, but suppress the close-by ones that are non-maximal

In this example there was only one object class (car).

If the initial detection contains multiple object classes, **non-max suppression should be carried out separately** for each detected object class.

Anchor boxes

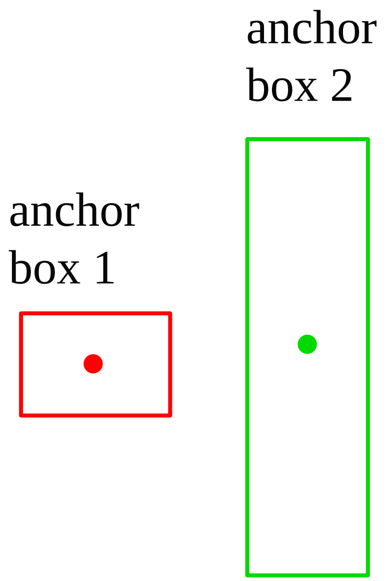
- So far, each grid cell detects only one object. What if a grid cell need to detect multiple objects?



The central point of the **person** (green) and the central point of the **bag** (red) are both located in the same grid cell. For the cell there is only one label, so we cannot output two predictions.

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

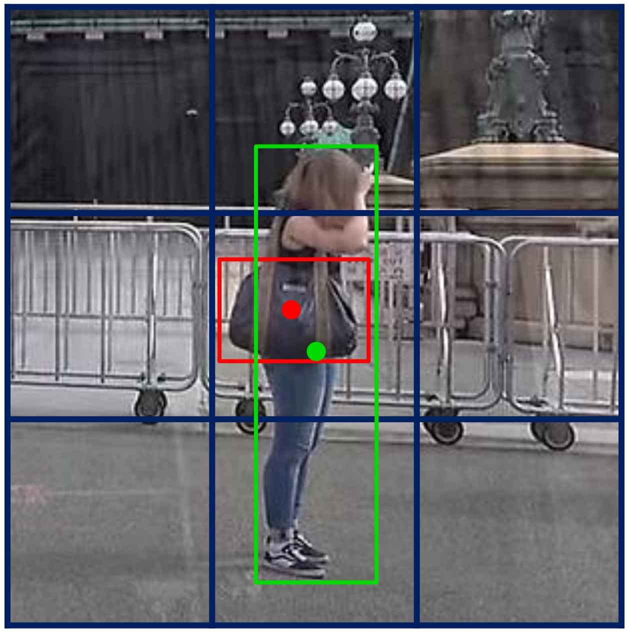
c_1	– person
c_2	– bag
c_3	– backpack



According to YOLO, we can use **anchor boxes** **(casete de ancorare):** **predefined shapes**.
We can associate one prediction for each anchor box.
In our example, for the same cell we will associate two predictions, for two anchor boxes.

In general, in applications, more anchor boxes will be used (5 or even 10)

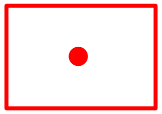
Anchor boxes



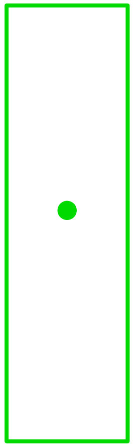
3 x 3 grid cells

c_1 – person
c_2 – bag
c_3 – backpack

anchor box 1



anchor box 2

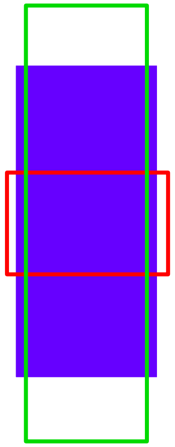


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The **output** of the ConvNet in this case will be:
3 x 3 x (2 x 8), meaning **3 x 3 x 16**

In the training set, each **object** will be assigned to the bounding box with maximum IoU

person – anchor box 2
bag – anchor box 1



IoU = 0.28
IoU = 0.75

If a cell contain only a person

Anchor boxes

What about if the number of objects in one grid cell > number of anchor boxes?
What about if a grid cell contains more objects with the same bounding box shape?

- this are not explicitly treated / let’s hope it won’t happen

YoloV2 (5 anchors) and YoloV3 (9 anchors)

Thus, the network should not predict the final size of the object but should only adjust the size of the nearest anchor to the size of the object.

In Yolo v3 anchors (width, height) - are sizes of objects on the image that resized to the network size (width =   height =   in the cfg-file).

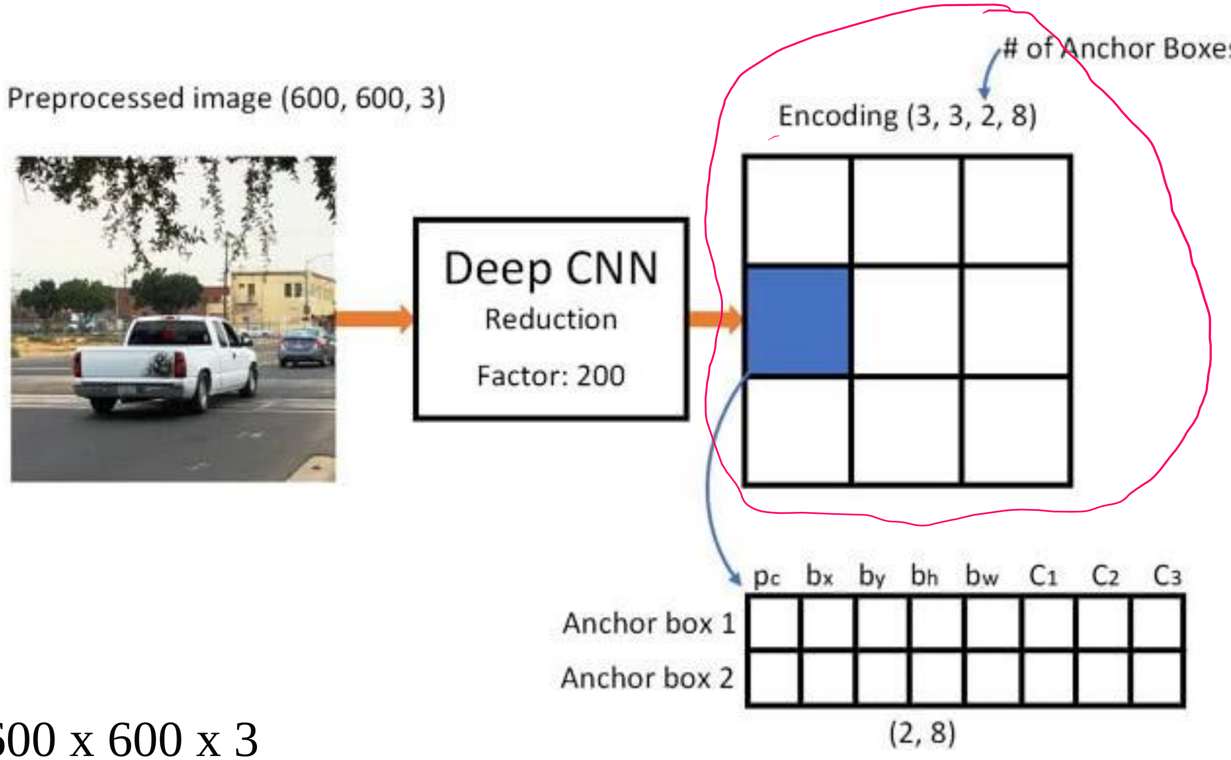
In Yolo v2 anchors (width, height) - are sizes of objects relative to the final feature map (32 times smaller than in Yolo v3 for default cfg-files).

YOLO – Real Time Object Detection

YOLO output volume

Training the YOLO model

3 object classes
2 anchor boxes
3 x 3 grid cell



Training set input: image 600 x 600 x 3

Training set target: volume 3 x 3 x 16 [3 x 3 x 2 x 8, 3 x 3 x 2 x (1+4+3)]

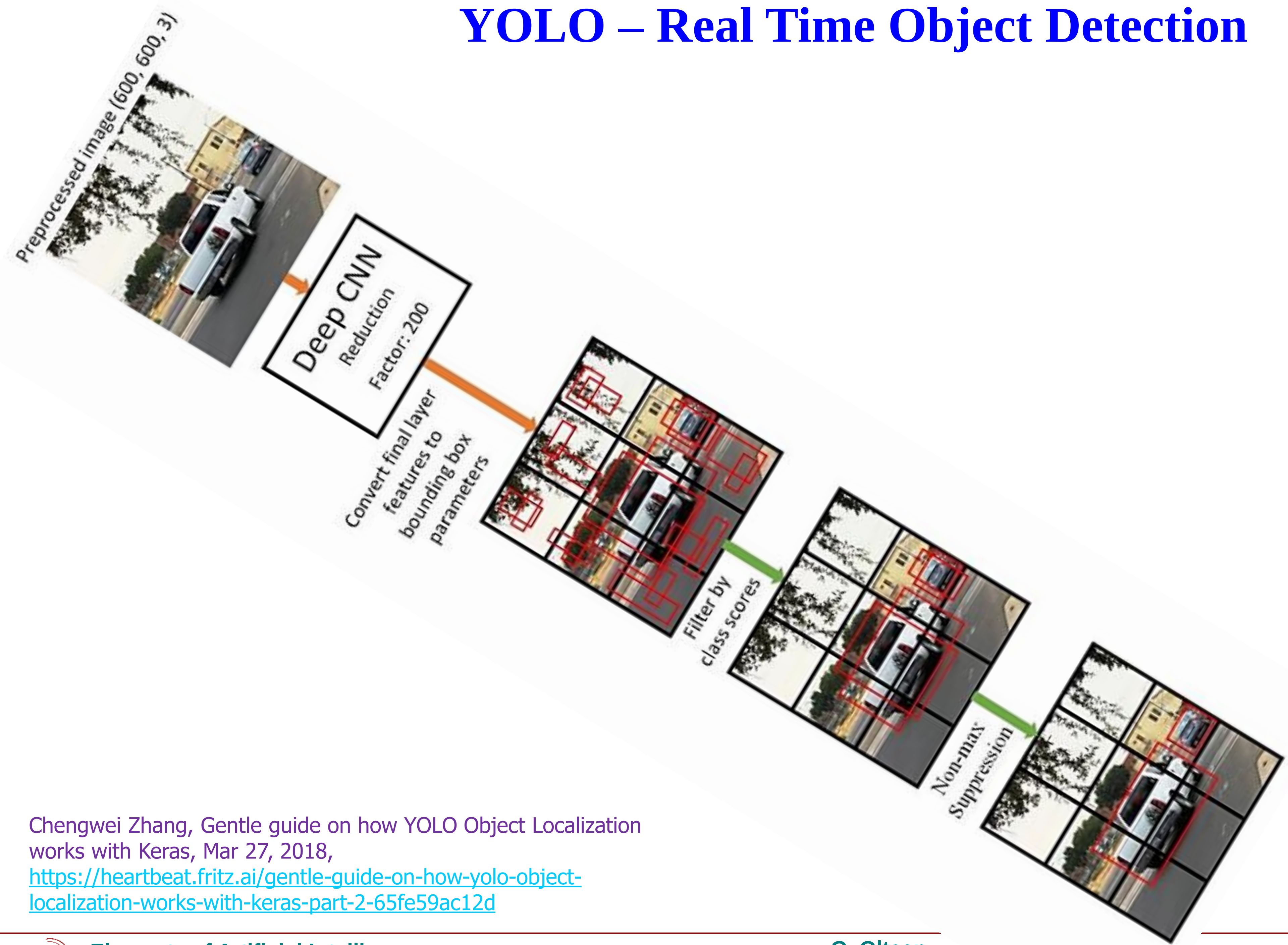
If 10 classes and 5 anchor boxes, and 19 x 19 grid cell

Training set input: image 600 x 600 x 3

Training set target: volume 19 x 19 x 75 [19 x 19 x 5 x 15, 19 x 19 x 5 x (1+4+10)

Chengwei Zhang, Gentle guide on how YOLO Object Localization works with Keras, Mar 27, 2018, https://heartbeat.fritz.ai/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2-65fe59ac12d

YOLO – Real Time Object Detection


Preprocessed image (600, 600, 3)
Deep CNN
Reduction
Factor: 200
Convert final layer features to bounding box parameters
Filter by class scores
Non-max Suppression

Chengwei Zhang, Gentle guide on how YOLO Object Localization works with Keras, Mar 27, 2018, https://heartbeat.fritz.ai/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2-65fe59ac12d

YOLOv3 in the CLOUD : Install and Train Custom Object Detector (FREE GPU)

Learn how get YOLOv3 object detection running in the cloud with Google Colab. Walk-through the steps to run yolov3 with darknet detections in the cloud and how to train your very own custom object detector. ALL WITH FREE GPU!

This tutorial covers it all! #yolov3 #objectdetection #cloud

GOOGLE COLAB NOTEBOOK: https://colab.research.google.com/dri...

This video cover:

1. Setting up Google Colab as a cloud VM with Free GPU.
2. Commands to get Darknet with YOLOv3 weights installed and running.
3. YOLOv3 pretrained coco model detections in the Cloud.
4. Configuration for Custom YOLOv3 Training in the Cloud.
5. Training Custom YOLOv3 Object Detector in the Cloud.

YOLOv4 in the CLOUD: Install and Run Object Detector (FREE GPU)

Learn how to get YOLOv4 Object Detection running in the Cloud with Google Colab. YOLOv4 is brand new and boasts many performance and speed upgrades over it's older version, YOLOv3. YOLOv4 is one of the world's fastest and most accurate object detection systems. Walk-through the steps to run yolov4 with darknet detections in the cloud and harness it's vast power and speed. ALL WITH FREE GPU! This tutorial covers it all!

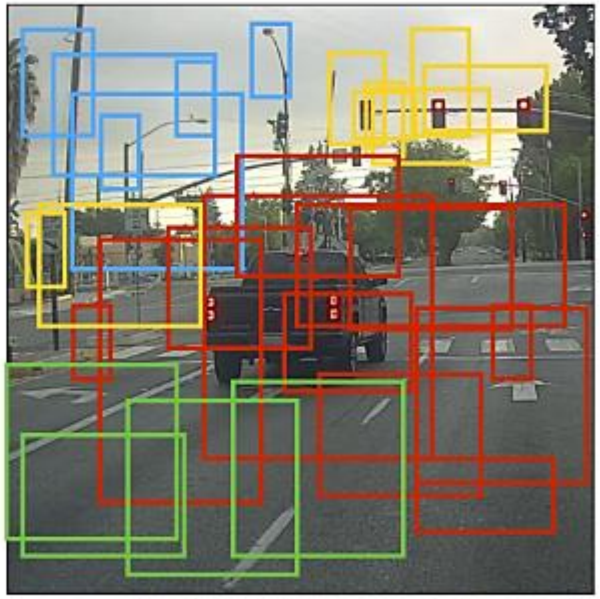
THE GOOGLE COLAB NOTEBOOK: https://colab.research.google.com/dri...
In this video I Cover:

1. Setting up Google Colab as a Cloud VM with Free GPU.
2. Commands to Build Darknet with YOLOv4 weights installed.
3. Running YOLOv4 pre-trained coco model detections in the Cloud.
4. Performing YOLOv4 detections on video in the Cloud.
5. How to run Custom YOLOv4 commands with various flags.
6. Performing YOLOv4 detections on multiple images at once.
7. Saving YOLOv4 detections to JSON and Text Files.

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

Visualizing bounding boxes

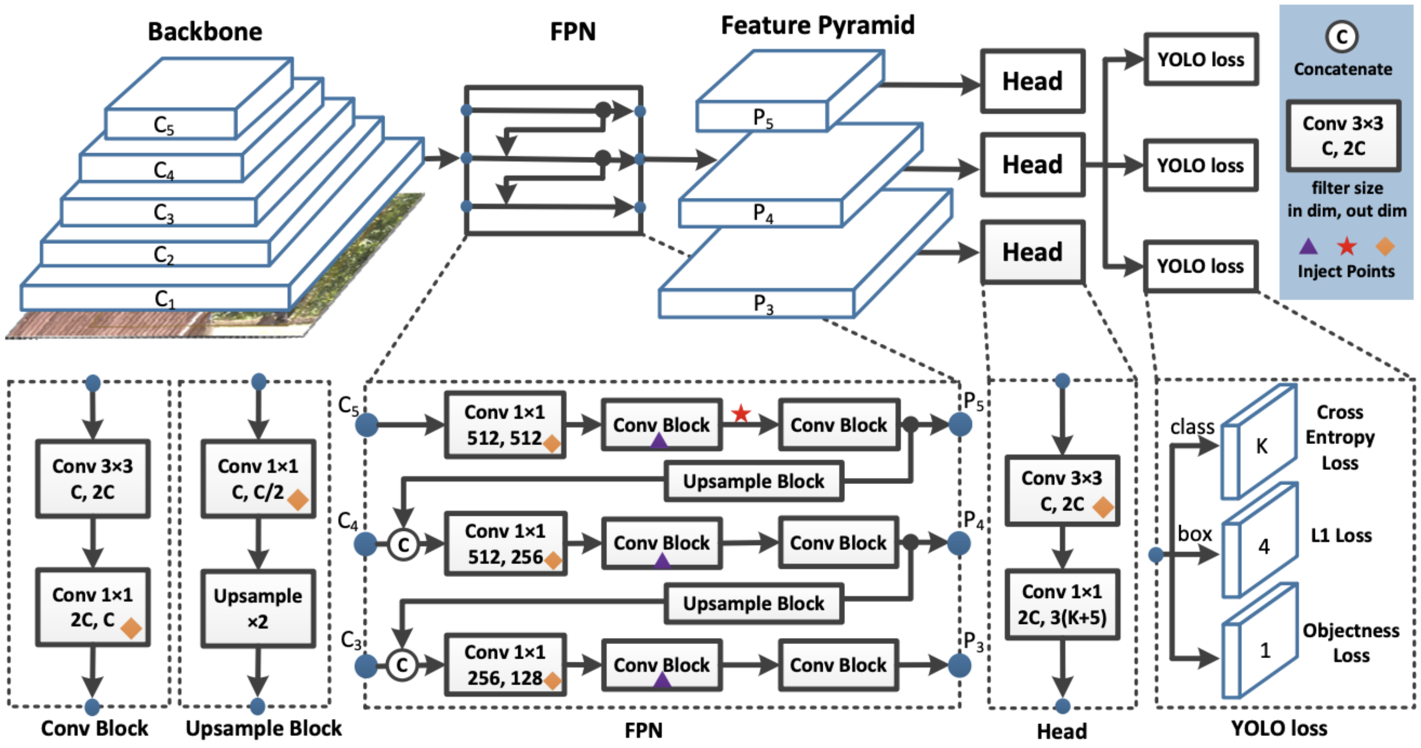
Another way to visualize YOLO's output is to plot the bounding boxes that it outputs. Doing that results in a visualization like this:



<u>Figure 6</u> : Each cell gives you 5 boxes. In total, the model predicts: 19x19x5 = 1805 boxes just by looking once at the image (one forward pass through the network)! Different colors denote different classes.

Jacob Solawetz, PP-YOLO Surpasses YOLOv4 — Object Detection Advances, 4 aug 2020, https://towardsdatascience.com/pp-yolo-surpasses-yolov4-object-detection-advances-1efc2692aa62

Anatomy of the YOLO Detector: it is broken into three main pieces.



YOLO Backbone — The YOLO backbone is a convolutional neural network that pools image pixels to form features at different granularities. The Backbone is typically pretrained on a classification dataset, typically ImageNet.

YOLO Neck — The YOLO neck (FPN is chosen above) combines and mixes the ConvNet layer representations before passing on to the prediction head.

YOLO Head — This is the part of the network that makes the bounding box and class prediction. It is guided by the three YOLO loss functions for class, box, and objectness.