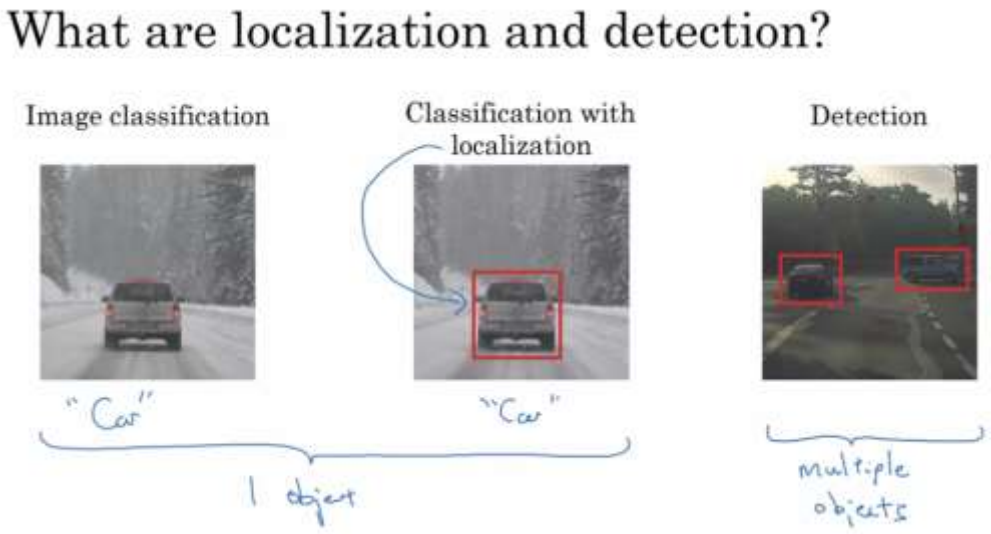
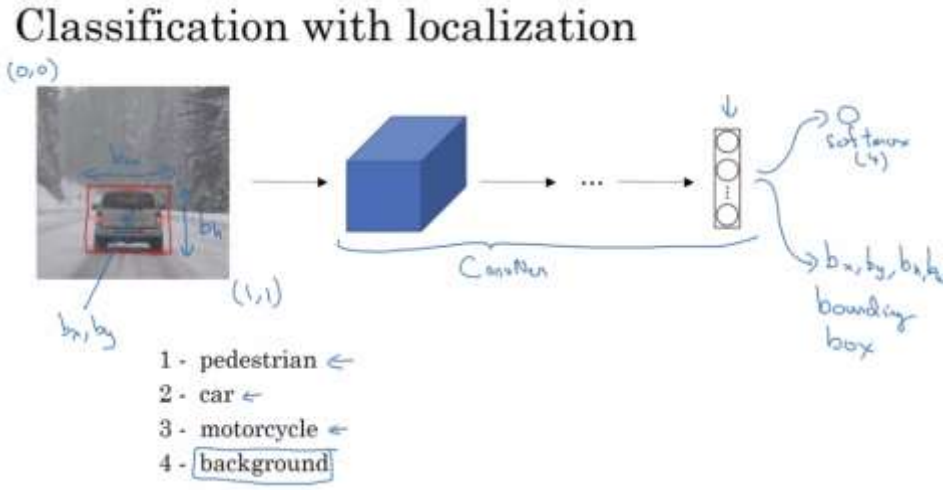
Convolutional Neural Networks

Week 3





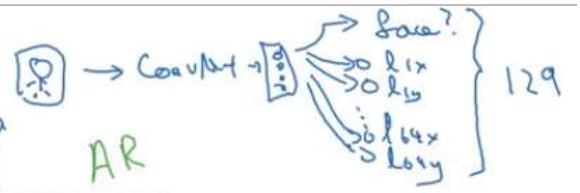
This is the form of output Y we will use:

$\mathcal{L}(\hat{y}, y) =$

$$\begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

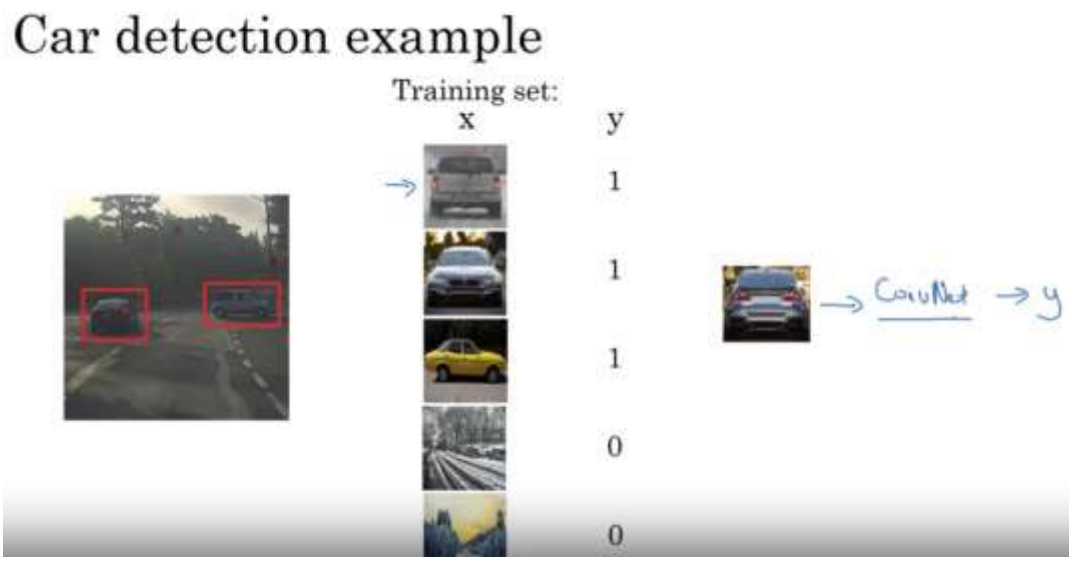
$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ → is there an object?

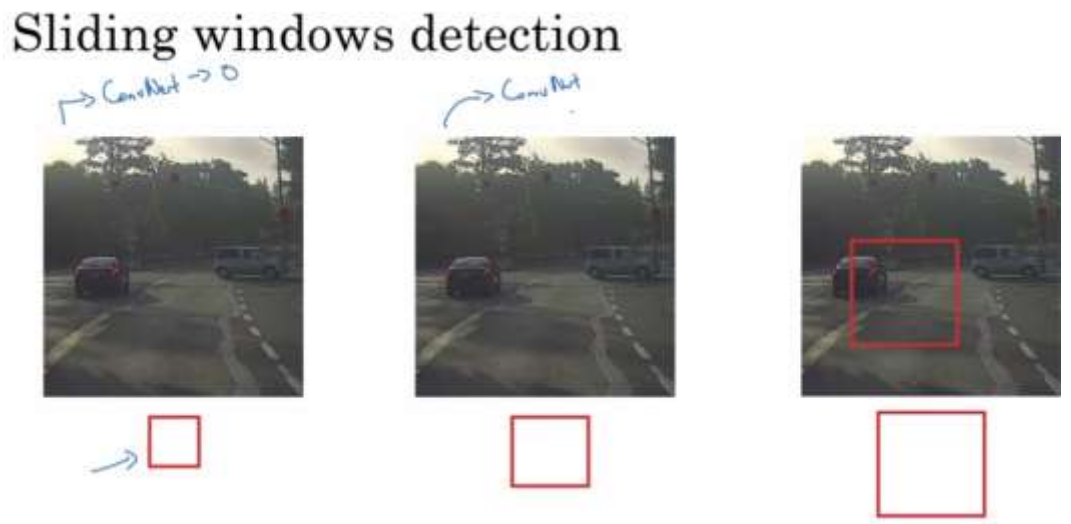
Landmark detection



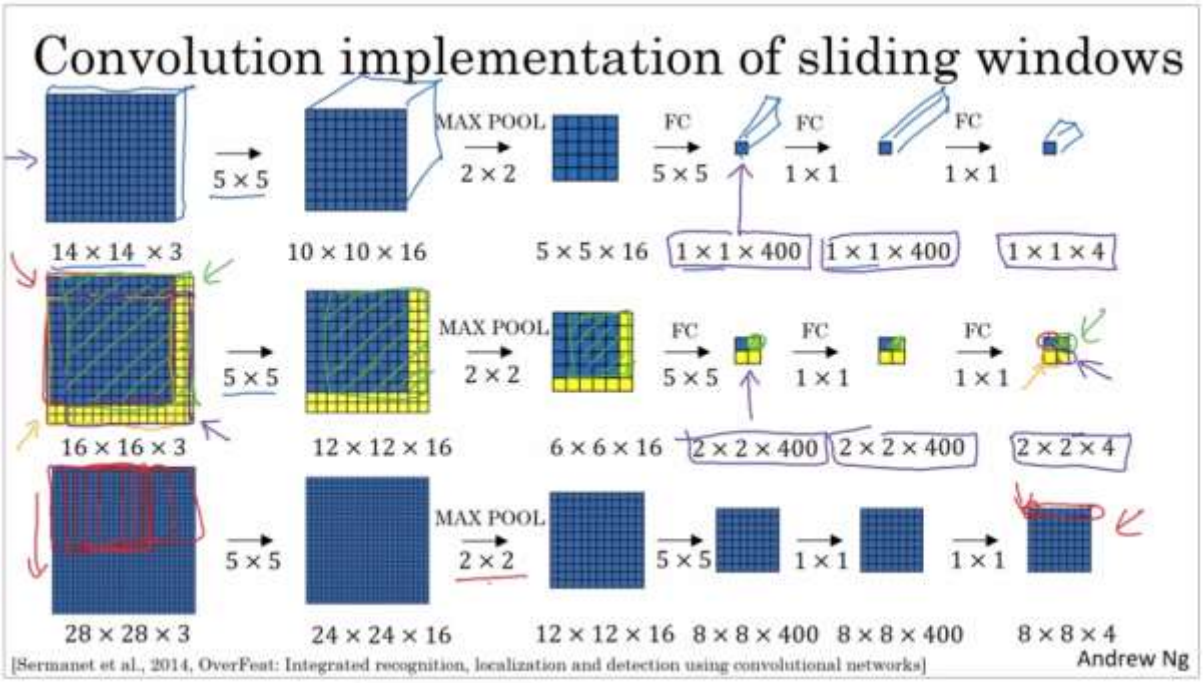
b_x, b_y, b_h, b_w

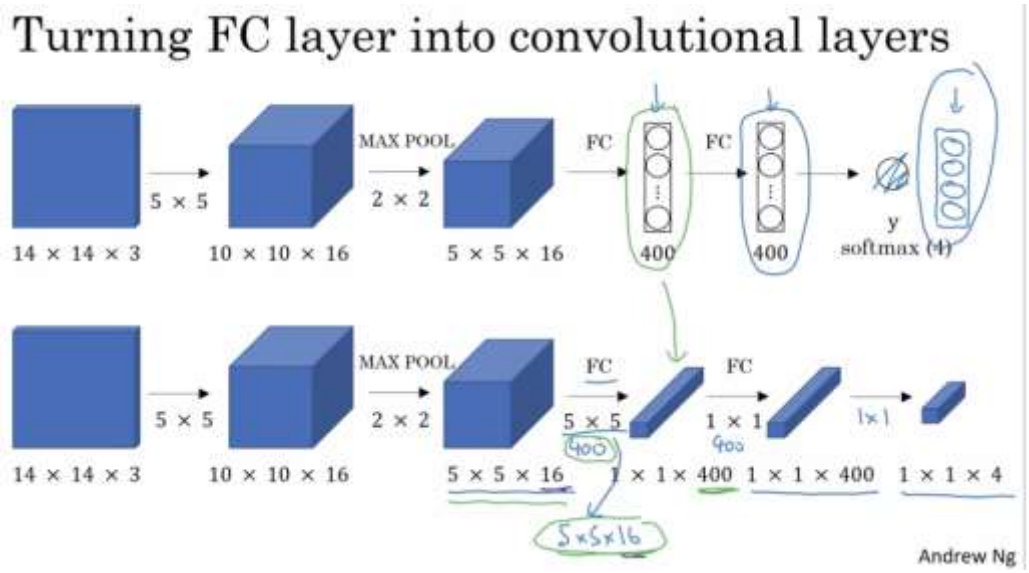
$l_{1x}, l_{1y},$
$l_{2x}, l_{2y},$
$l_{3x}, l_{3y},$
l_{4x}, l_{4y}
$\vdots$
l_{64}, l_{64y}



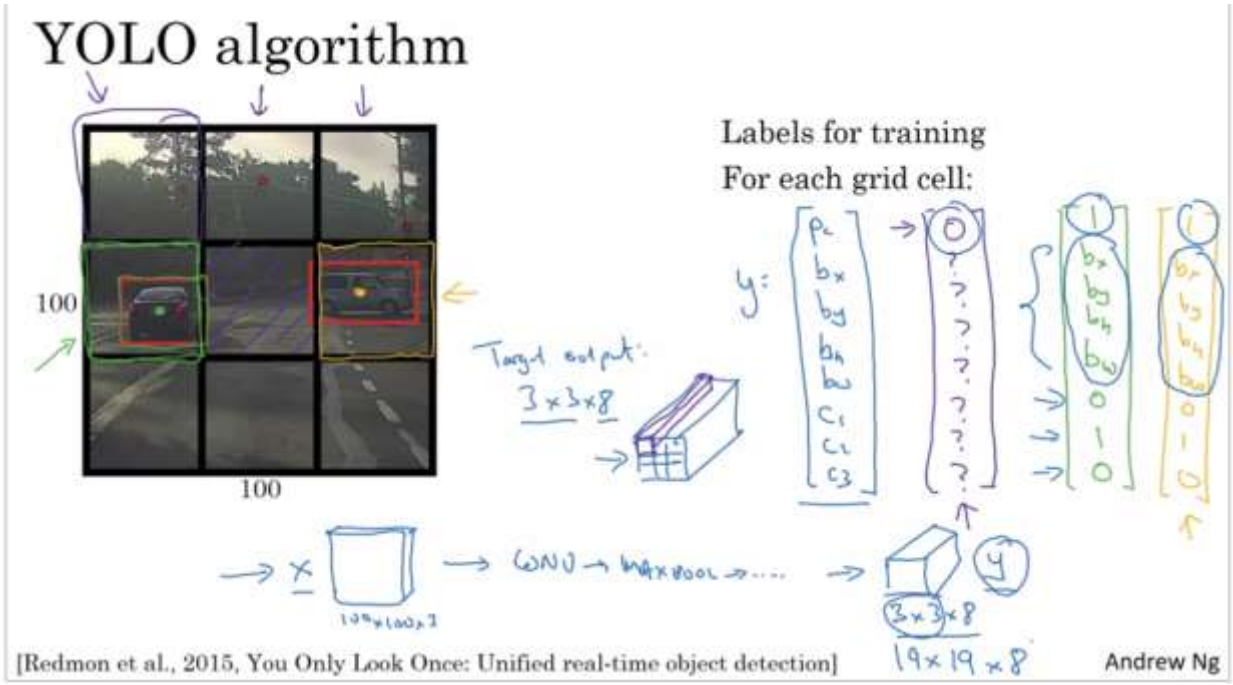


Unfortunately, this algorithm is too slow to be used in practice, as every sliding window has to do a forward propagation in the NN.



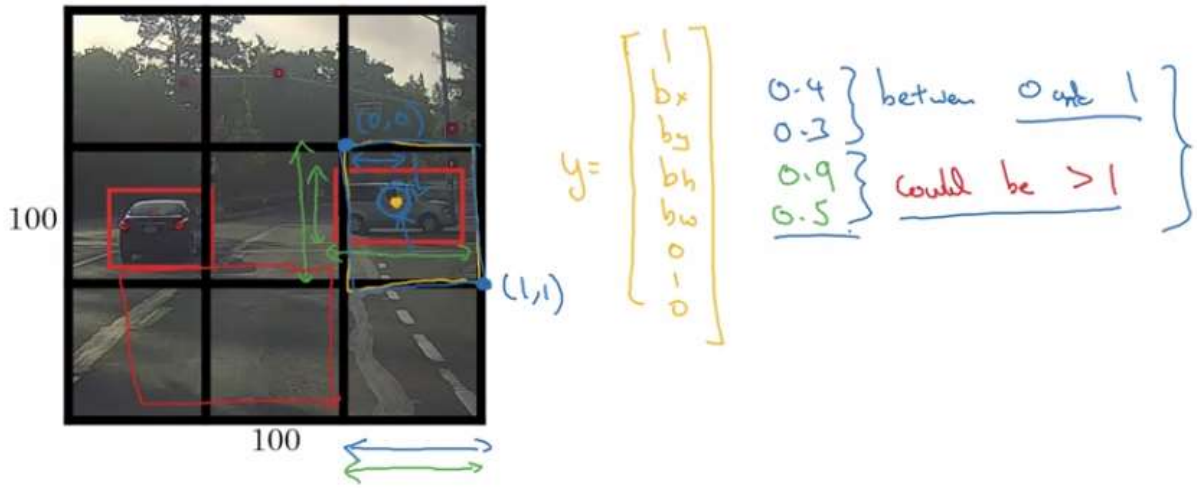


Although this approach is much more computationally efficient, it still has a problem: the "sliding window" simulated with the convolutional layer does not always fit perfectly the image.

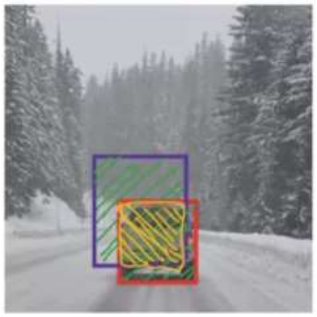


[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Specify the bounding boxes



Evaluating object localization



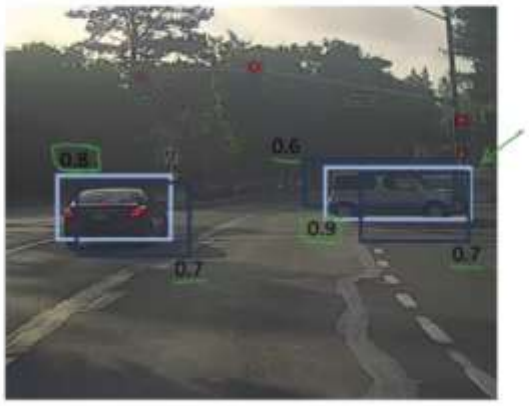
Intersection over Union (IoU)

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

"Correct" if IoU $\geq$ 0.5

0.6

Non-max suppression example



p_c

Non-max suppression algorithm

19× 19

Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Discard all boxes with $p_c \leq 0.6$

While there are any remaining boxes:

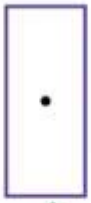
- Pick the box with the largest p_c
  Output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step

Andrew Ng

Overlapping objects:

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

Anchor box 1:



Anchor box 2:

$y =$ $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ Anchor box 1 / Anchor box 2

Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

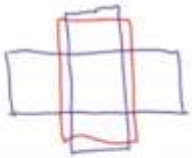
Andrew Ng

Anchor box algorithm

Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y:
3×3×8



With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

(grid cell, anchor box)

Output y:
3×3×16
3×3×2×8

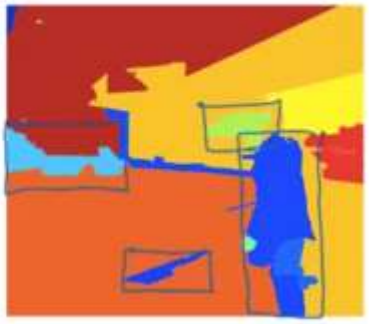
Andrew Ng

Of course, there is a very slim chance of collision on the same grid cell of objects of the same anchor box; but as I said, this happens very rare in practice.

Outputting the non-max supressed outputs

- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Region proposal: R-CNN



The output in the right is obtained using an algorithm called “object segmentation”.

Faster algorithms

R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box.

Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.

Faster R-CNN: Use convolutional network to propose regions.

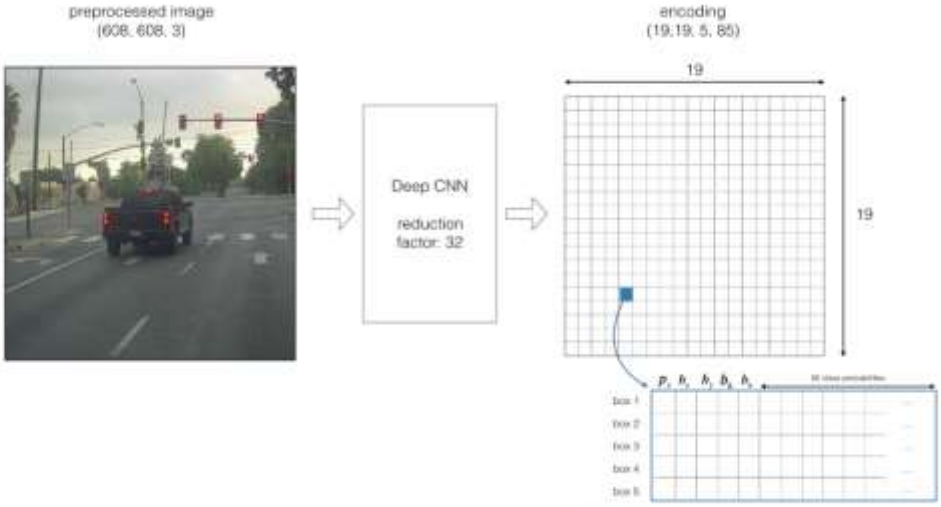
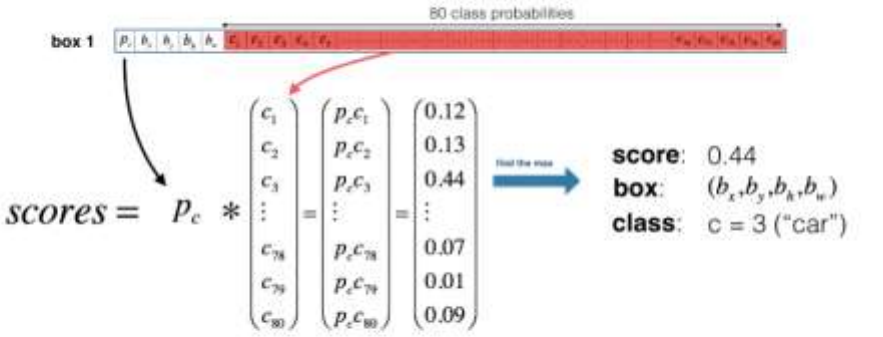


Figure 2 : **Encoding architecture for YOLO**

Now, for each box (of each cell) we will compute the following element-wise product and extract a probability that the box contains a certain class.
The class score is $score_{c,i} = p_c \times c_i$: the probability that there is an object p_c times the probability that the object is a certain class c_i.



the box (b_x, b_y, b_h, b_w) has detected c = 3 (“car”) with probability score: 0.44

Figure 4 : **Find the class detected by each box**

Here's one way to visualize what YOLO is predicting on an image:

- For each of the 19x19 grid cells, find the maximum of the probability scores (taking a max across the 80 classes, one maximum for each of the 5 anchor boxes).
- Color that grid cell according to what object that grid cell considers the most likely.

Doing this results in this picture:

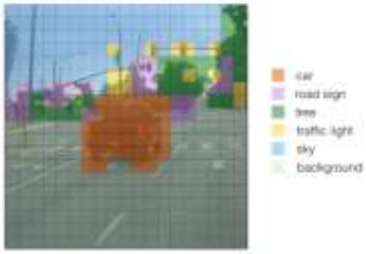


Figure 5 : Each one of the 19x19 grid cells is colored according to which class has the largest predicted probability in that cell.

Note that this visualization isn't a core part of the YOLO algorithm itself for making predictions; it's just a nice way of visualizing an intermediate result of the algorithm.

Reference documentation

- tf.image.non_max_suppression()

```
tf.image.non_max_suppression(
  boxes,
  scores,
  max_output_size,
  iou_threshold=0.5,
  name=None
)
```

  Note that in the version of tensorflow used here, there is no parameter `score_threshold` (it's shown in the documentation for the latest version) so trying to set this value will result in an error message: *got an unexpected keyword argument 'score_threshold*.
- K.gather()
  Even though the documentation shows `tf.keras.backend.gather()`, you can use `keras.gather()`.

```
keras.gather(
  reference,
  indices
)
```