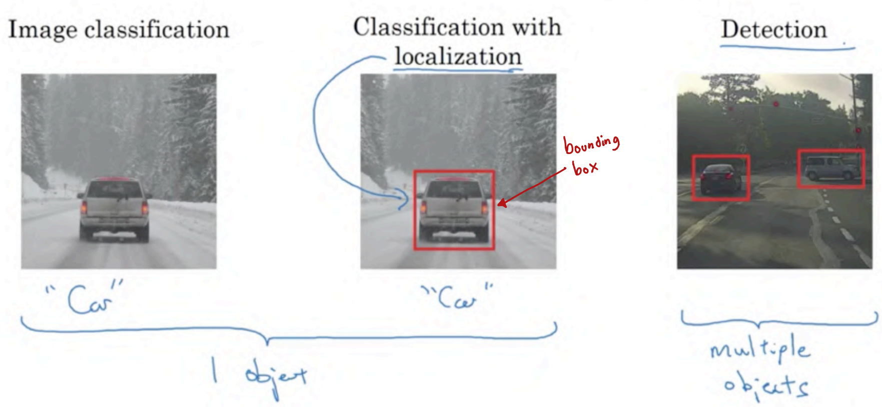
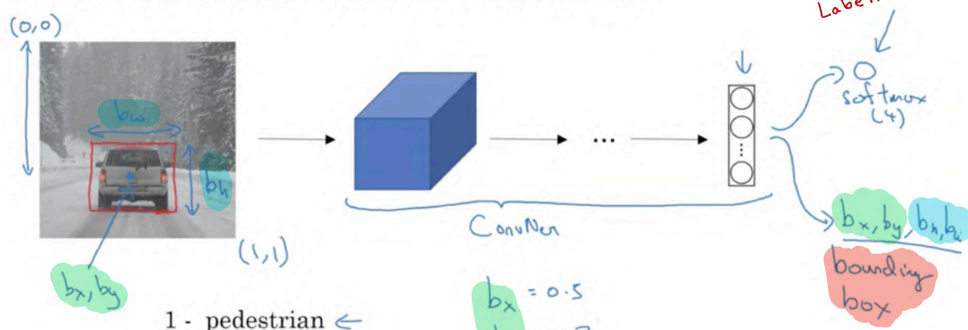
What are localization and detection?



Andrew Ng

Classification with localization



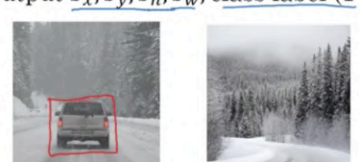
1 - pedestrian
2 - car
3 - motorcycle
4 - $\boxed{\text{background}}$

$b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

Defining the target label y

1 - pedestrian c_1
2 - car c_2
3 - motorcycle c_3
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)



loss

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

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

logistic regression loss

is there any object? 1 or 0

squared error

loglikelihood

$x =$

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

(x, y)

p_c

"don't care"

None

Andrew Ng

Landmark detection

→ ConvNet → face? l_{1x} l_{1y} … l_{64x} l_{64y} } 129

ConvNet

AR

b_x, b_y, b_h, b_w

$l_{1x}, l_{1y},$
$l_{2x}, l_{2y},$
$l_{3x}, l_{3y},$
l_{4x}, l_{4y}
⋮
l_{64x}, l_{64y}
} X, y

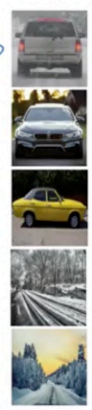
e.g. Find eyes → Points on face →

$l_{1x}, l_{1y},$
⋮
l_{32x}, l_{32y}

Andrew Ng

Car detection example

Training set:

x	y
	1
	1
	1
	0
	0

 → ConvNet → y

Andrew Ng

Sliding windows detection

→ ConvNet → 0

Sliding windows detection

→ ConvNet → 0

Sliding windows detection

→ ConvNet → 0

Pass crops to have it classify 0 or 1

Sliding windows detection

Sliding windows detection

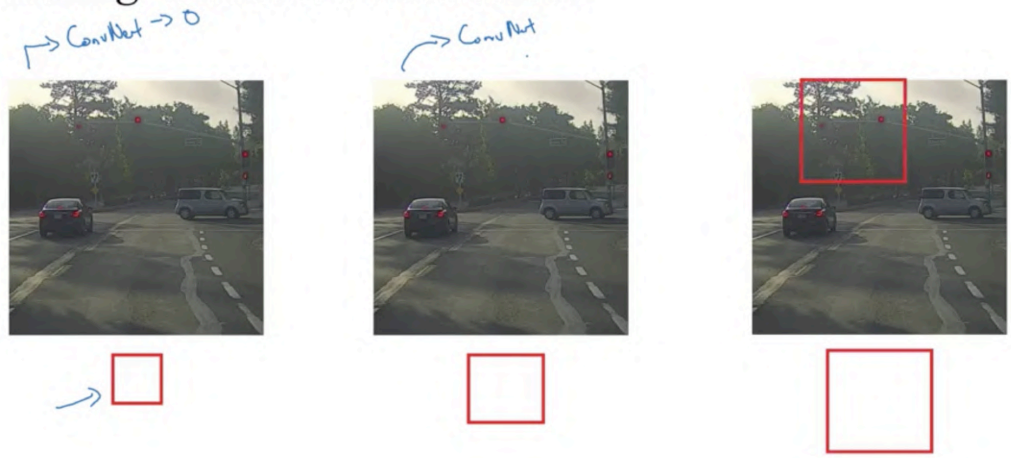
Sliding windows detection

Sliding windows detection

Sliding windows detection

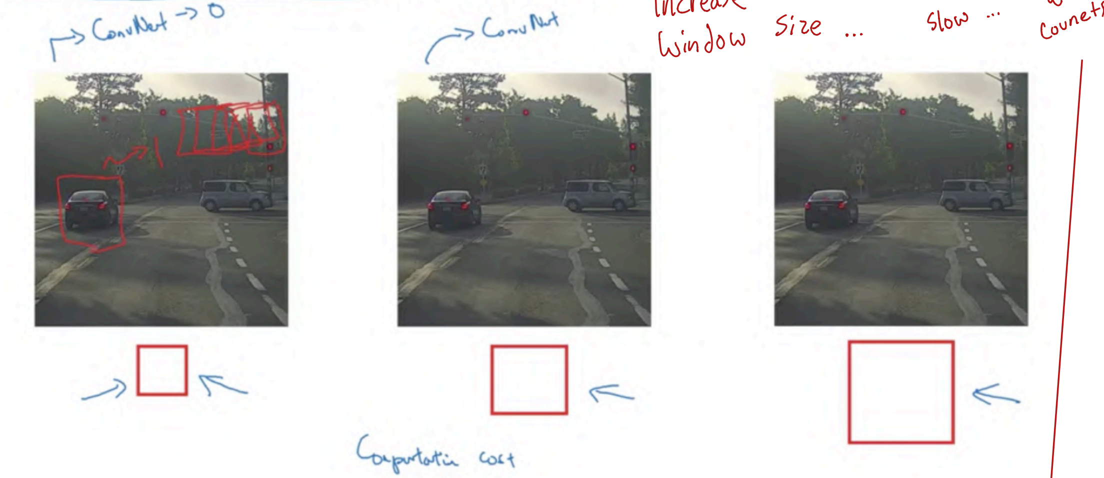
Sliding windows detection

Sliding windows detection

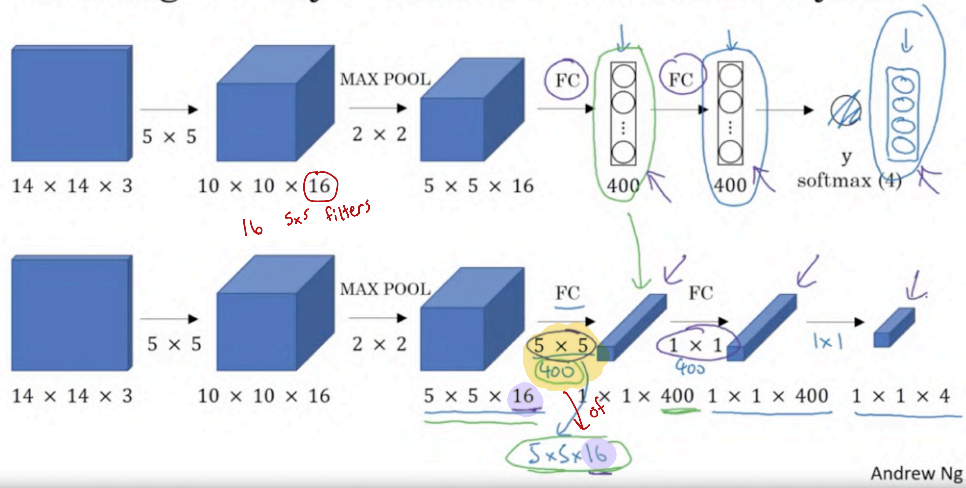


Sliding windows detection

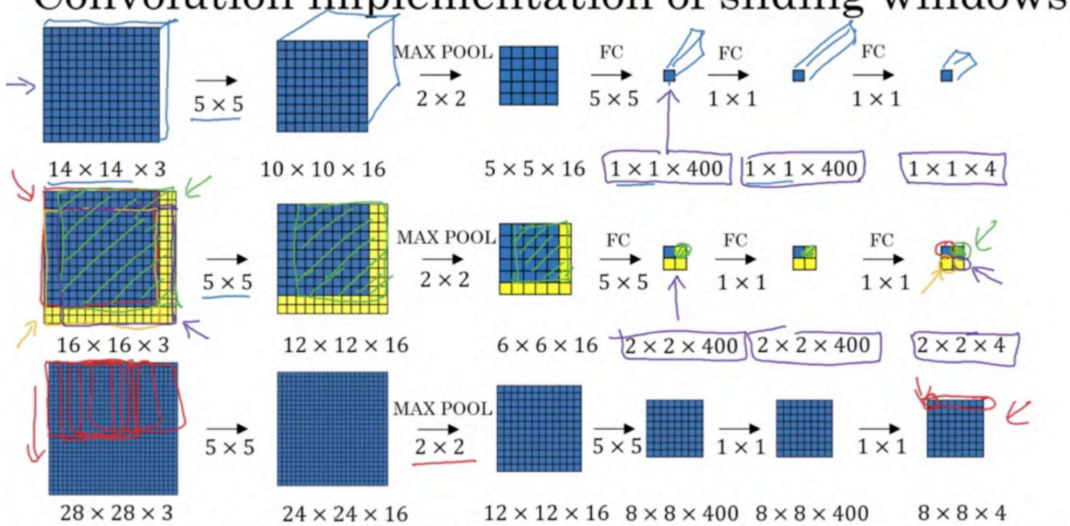
Sliding windows detection



Turning FC layer into convolutional layers



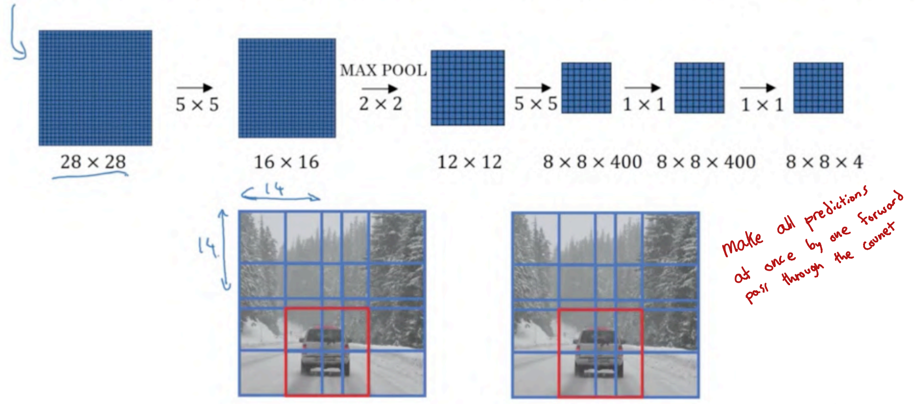
Convolution implementation of sliding windows



[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Andrew Ng

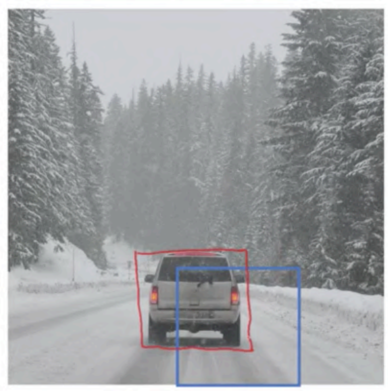
Convolution implementation of sliding windows



Make all predictions at once by one forward pass through the convnet

Andrew Ng

Output accurate bounding boxes



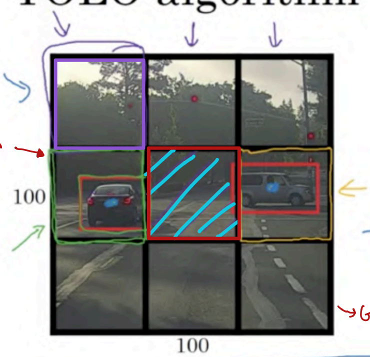
Andrew Ng

YOLO algorithm

This image has two objects. And what the YOLO algorithm does is it takes the midpoint of reach of the two objects and then assigns the object to the grid cell containing the midpoint. So the left car is assigned to this grid cell, and the car on the right, which is this midpoint, is assigned to this grid cell. And so even though the central grid cell has some parts of both cars, we'll pretend the central grid cell has no interesting object so that the central grid cell the class label Y also looks like this vector with no object, and so the first component PC, and then the rest are don't cares. Whereas for this cell, this cell that I have circled in green on the left, the target label Y would be as follows.

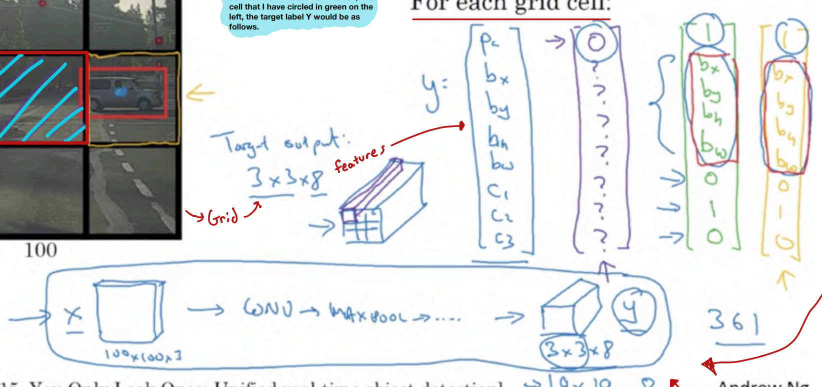
So the advantage of this algorithm is that the neural network outputs precise bounding boxes as follows. So at test time, what you do is you feed an input image X and run forward prop until you get this output Y. And then for each of the nine outputs of each of the 3 by 3 positions in which of the output, you can then just read off 1 or 0. Is there an object associated with that one of the nine positions? And that there is an object, what object it is, and where is the bounding box for the object in that grid cell? And so long as you don't have more than one object in each grid cell, this algorithm should work okay. And the problem of having multiple objects within the grid cell is something we'll address later. Of use a relatively small 3 by 3 grid, in practice, you might use a much finer, grid maybe 19 by 19. So you end up with 19 by 19 by 8, and that also makes your grid much finer. It reduces the chance that there are multiple objects assigned to the same grid cell. And just as a reminder, the way you assign an object to grid cell as you look at the midpoint of an object and then you assign that object to whichever one grid cell contains the midpoint of the object. So each object, even if the objects spends multiple grid cells, that object is assigned only to one of the nine grid cells, or one of the 3 by 3, or one of the 19 by 19 grid cells.

Apply Image Classification and localization to each of the 9 grids



Labels for training

For each grid cell:



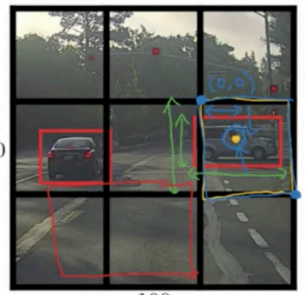
[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Andrew Ng

Each object only assigned to 1

grid cell

Specify the bounding boxes



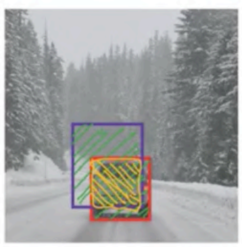
$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

0.4, 0.3 } between 0 and 1

0.9, 0.5 } could be > 1

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

Evaluating object localization



Intersection over Union (IoU)

$$= \frac{\text{size of } \text{(yellow)}}{\text{size of } \text{(green)}}$$

"Correct" if IoU ≥ $\boxed{0.5}$ ← rarely below 0.5

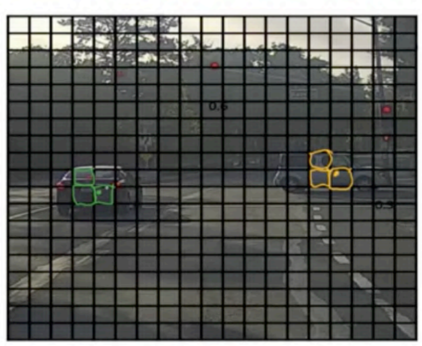
0.6 ←

More generally, IoU is a measure of the overlap between two bounding boxes.

Non-max suppression example

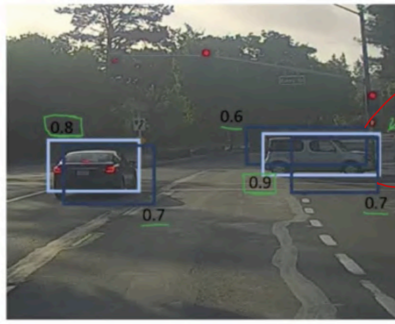
Since you're running the image classification and localization algorithm on every grid cell, on 361 grid cells, it's possible that many of them will raise their hand and say, "My Pc, my chance of thinking I have an object in it is large." Rather than just having two of the grid cells out of the 19 squared or 361 think they have detected an object. So, when you run your algorithm, you might end up with multiple detections of each object. So, what non-max suppression does, is it cleans up these detections. So they end up with just one detection per car, rather than multiple detections per car. So concretely, what it does, is it first looks at the probabilities associated with each of these detections.

let's just say is Pc with the probability of a detection. And it first takes the largest one, which in this case is 0.9 and says, "That's my most confident detection, so let's highlight that and just say I found the car there." Having done that the non-max suppression part then looks at all of the remaining rectangles and all the ones with a high overlap, with a high IOU, with this one that you've just output will get suppressed.



19x19

Non-max suppression example

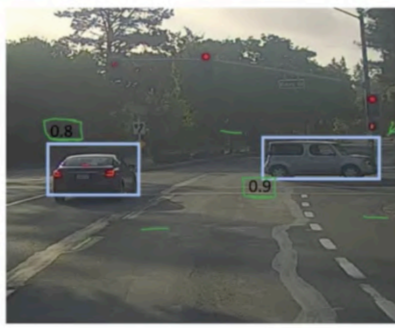


Suppress

Suppress

p_c

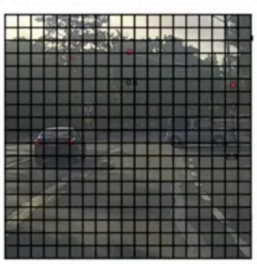
Non-max suppression example



p_c

Non-max suppression algorithm

If you actually tried to detect three objects say pedestrians, cars, and motorcycles, then the output vector will have three additional components. And it turns out, the right thing to do is to independently carry out non-max suppression three times, one on each of the outputs classes.



19× 19

assume only car detection for this example

Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$ $\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$

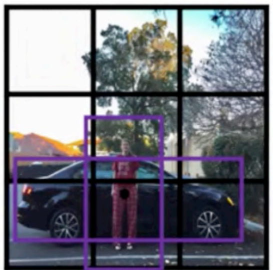
Discard all boxes with $p_c \leq 0.6$

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

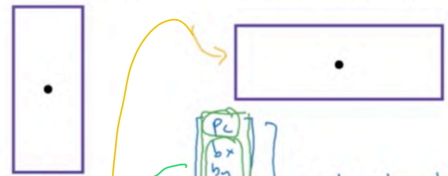
With the idea of anchor boxes, what you are going to do, is pre-define two different shapes called, anchor boxes or anchor box shapes. And what you are going to do is now, be able to associate two predictions with the two anchor boxes. And in general, you might use more anchor boxes,

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:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix} \begin{matrix} \text{anchor box 1} \\ \\ \text{anchor box 2} \end{matrix}$$

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

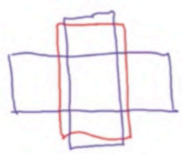
Andrew Ng

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.

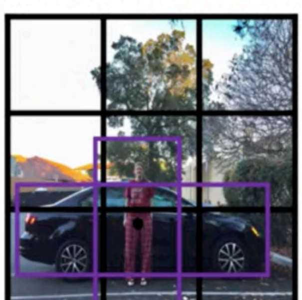
(grid cell, anchor box)

Output y:
3×3×16
3×3×2×8

\# of anchor boxes

Andrew Ng

Anchor box example



Anchor box 1: Anchor box 2:

Pedestrian

Car

car only?

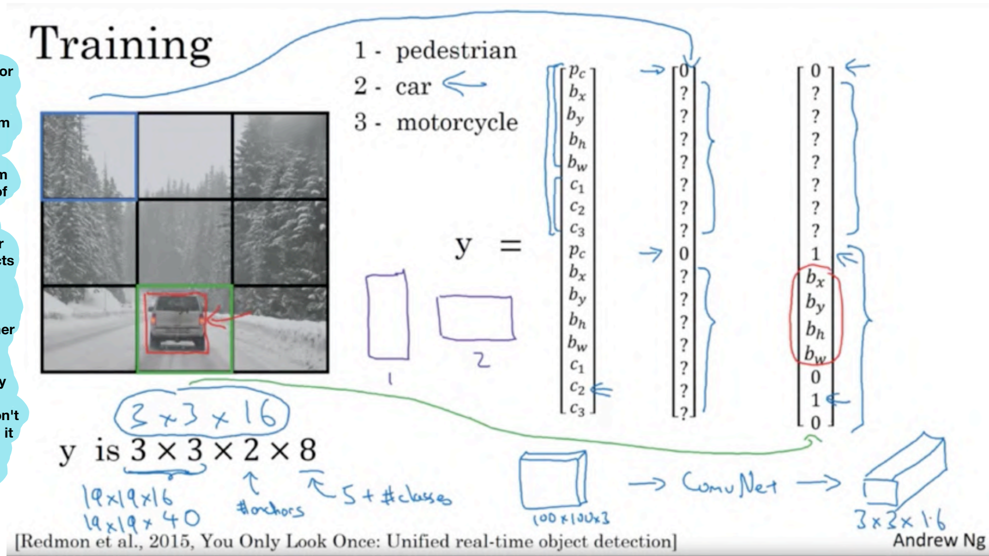
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \checkmark \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \text{anchor box 1} \\ \\ \text{anchor box 2} \end{matrix}$$

don't need anchor box for pedestrian

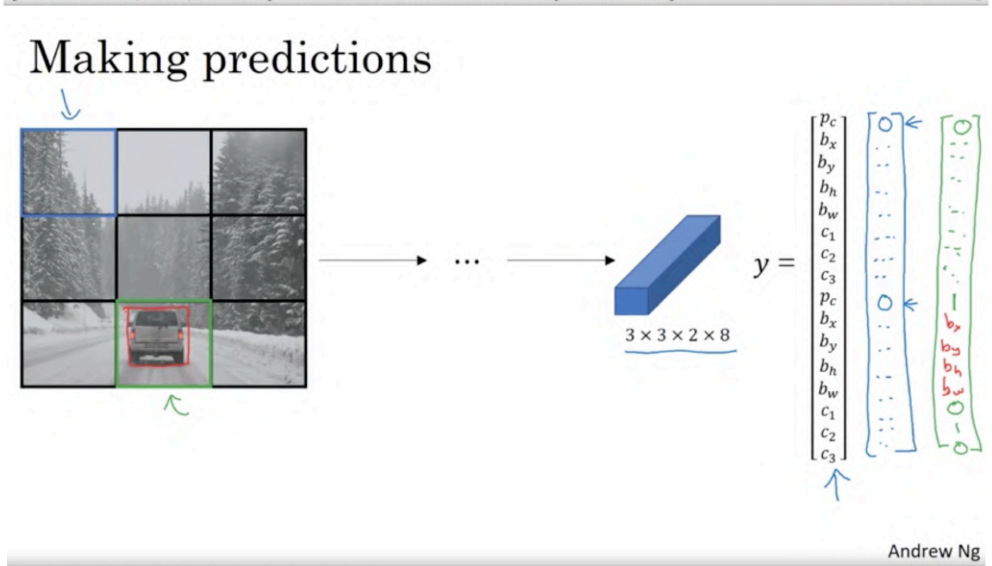
No object

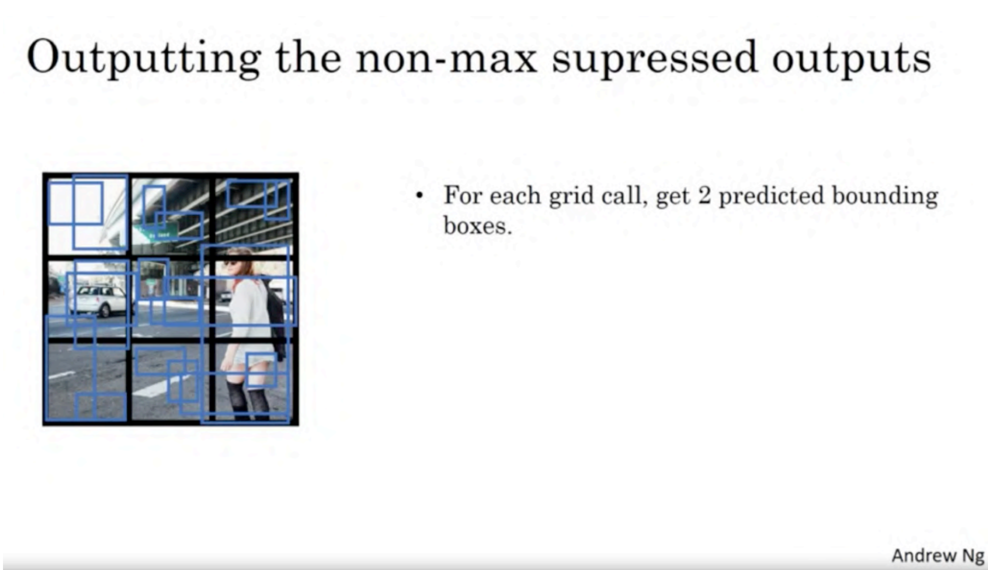
Andrew Ng

What if you have two anchor boxes but three objects in the same grid cell? That's one case that this algorithm doesn't handle well. Hopefully, it won't happen. But if it does, this algorithm doesn't have a great way of handling it. I will just influence some default tiebreaker for that case. Or what if you have two objects associated with the same grid cell, but both of them have the same anchor box shape? Again, that's another case that this algorithm doesn't handle well. If you influence some default way of tiebreaking if that happens, hopefully this won't happen with your data set, it won't happen much at all. And so, it shouldn't affect performance as much.



Even though I'd motivated anchor boxes as a way to deal with what happens if two objects appear in the same grid cell, in practice, that happens quite rarely, especially if you use a 19 by 19 rather than a 3 by 3 grid. The chance of two objects having the same midpoint rather these 361 cells, it does happen, but it doesn't happen that often. Maybe even better motivation or even better results that anchor boxes gives you is it allows your learning algorithm to specialize better. In particular, if your data set has some tall, skinny objects like pedestrians, and some white objects like cars, then this allows your learning algorithm to specialize so that some of the outputs can specialize in detecting white, fat objects like cars, and some of the output units can specialize in detecting tall, skinny objects like pedestrians. So finally, how do you choose the anchor boxes? And people used to just choose them by hand or choose maybe five or 10 anchor box shapes that spans a variety of shapes that seems to cover the types of objects you seem to detect. As a much more advanced version, just in the advance common for those of who have other knowledge in machine learning, and even better way to do this in one of the later YOLO research papers, is to use a K-means algorithm, to group together two types of objects shapes you tend to get. And then to use that to select a set of anchor boxes that this most stereotypically representative of the maybe multiple, of the maybe dozens of object causes you're trying to detect.





Outputting the non-max supressed outputs

- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.

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. — 3 times

Region proposal: $\boxed{\text{R}}$-CNN

Segmentation algorithm

~2,000

[Girshik et. al, 2013, Rich feature hierarchies for accurate object detection and semantic 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.

[Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]
[Girshik, 2015. Fast R-CNN]
[Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]