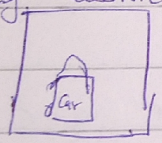
Week 3 Object detection Algorithms

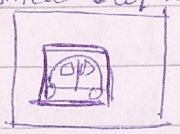
Object localization (To find local^n of object in the image)

Img. classificat^n



"car"

Classificat^n w/ localizat^n



"car"

Detection

multiple objects

Here, the o/p of the last softmax layer is a matrix:

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

- p_c → if object present or not
- b_x, b_y → midpoint of object
- b_h, b_w → height, width of the object box
- c_1, c_2, c_3 → To the class it belongs to.
  1 - pedestrian, 2 → car, 3 → ...

Loss function

if $p_c = 1$ (if obj present) $\rightarrow$

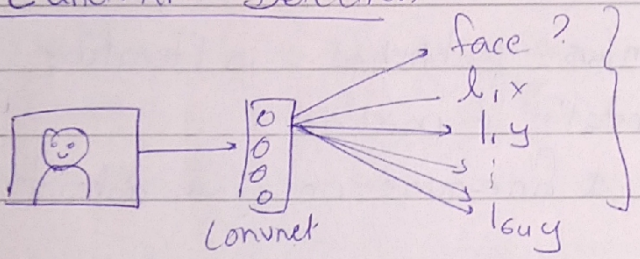
$$(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2$$

if $p_c = 0$ $\rightarrow$

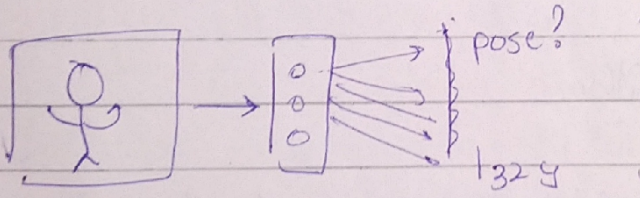
$$((\hat{y}_1 - y_1)^2)$$

don't care of rest indices if obj not present

Landmark Detection



129 features — each point giving important infor



33 features to determine pose of a person

Sliding Windao detection

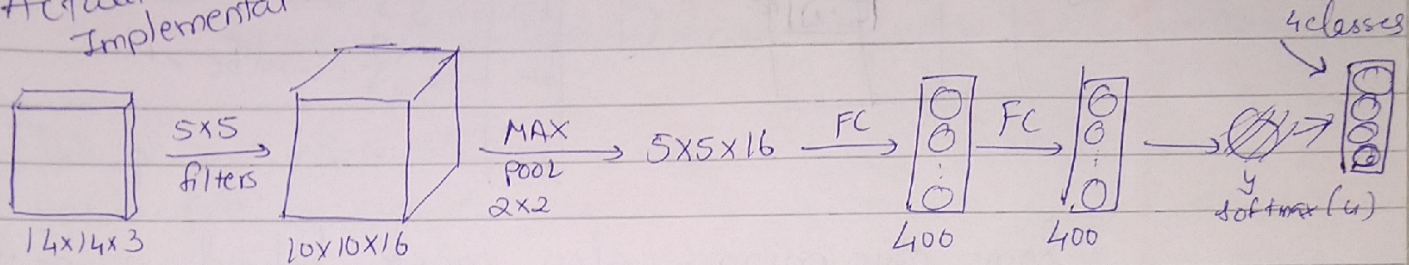
1) Train the modul to determine if there is a car or not in the image.
2) Next, take a box, you slide it throughout the image, and show each image to model to find, if there is a car in the box or not?

3) Then you resize the window and slide it again

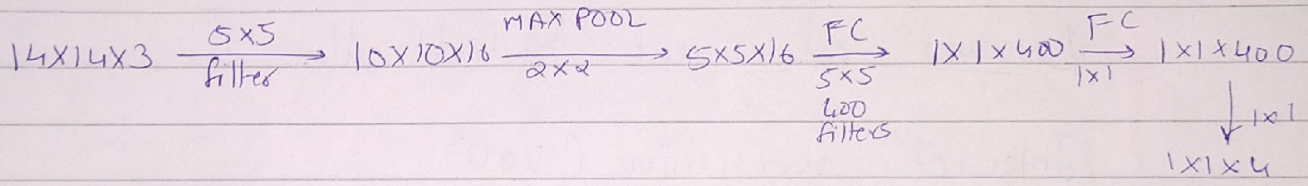
→ High Computatⁿ cost

→ But if implemented Convolutionally, it can be faster.

Actual Implementatⁿ



Convolutional Implementatⁿ

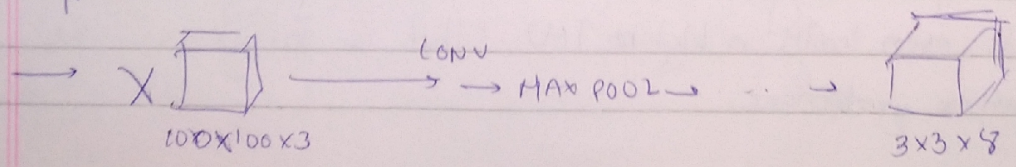


→ Here, the o/p of all the ~~sliding~~ sliding windows is expressed at the last region. The reason this is faster, is because each sliding window shares some computatⁿ. Also, instead of ~~adding~~ giving one window everytime to the model, it gives the whole image to the model.

→ The problem is, the bounding box size is inaccurate.

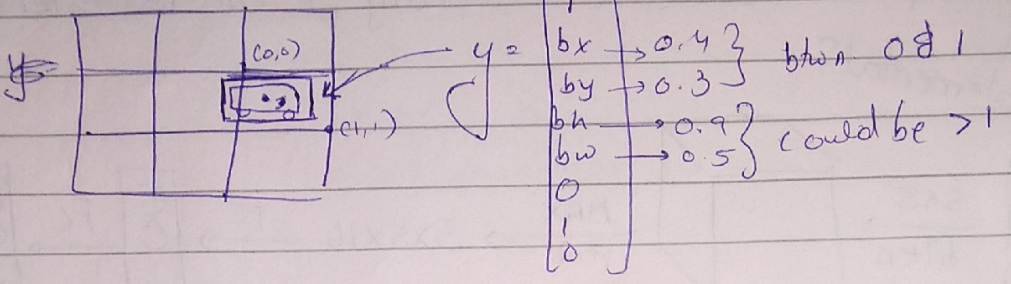
This can be solved by YOLO (You only Look Once)

The "image" is divided into 3×3 grid. Each grid has a 8 element vector. So, when we put the image in above implementatⁿ, we get o/p as 3×3×8



here, ~~any~~ In reality, we use a finer grid 19×19 instead of 3×3.

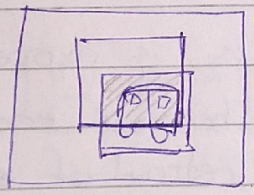
Specifying the bounding boxes in YOLO



Some other params also work behind giving the perfect box to the object

Evaluat^n of Object Detect^n Algos

Intersect^n over union (IoU)



$$= \frac{\text{Size of Intersection}}{\text{Size of Union}}$$

"correct" if $IoU \geq 0.5$

↑ IoU ↑ accuracy

To avoid your model to detect the same object again, you use "Non-Max Suppression"

When you detect an object multiple times, it has a P_c (probability of detect^n) associated with it.
The one with highest P_c are chosen, getting rid of other estimat^s even with a high IoU. That is, suppress the non-max predictions.

Algorithm:

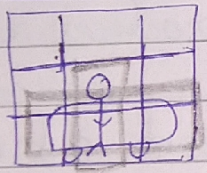
Input: image of 19x19x8

1) Discard all boxes with $P_c \leq 0.6$
2) while boxes left:
   - o/p the one with highest P_c as a predictn
   - Discard any remaining boxes w/ IoU ≥ 0.5 w/ the box o/p in previous step (discarding all other boxes of the same object)

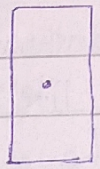
This process will be done for a single class, for multiple classes, you need to run it multiple times

When there are more than one object in a grid cell → Anchor Boxes

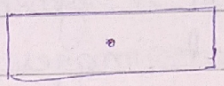
Anchor boxes



Anchor Box 1



Anchor Box 2



$y =$ [P_c, B_x, B_y, b_h, b_w, c_1, c_2, c_3, P_c, B_x, ..., c_3]

Anchor Box 1 | Anchor Box 2

We define two different sized anchor boxes to define each object

Previously:
Each object in training image is assigned to grid cell that contains that objects midpoint

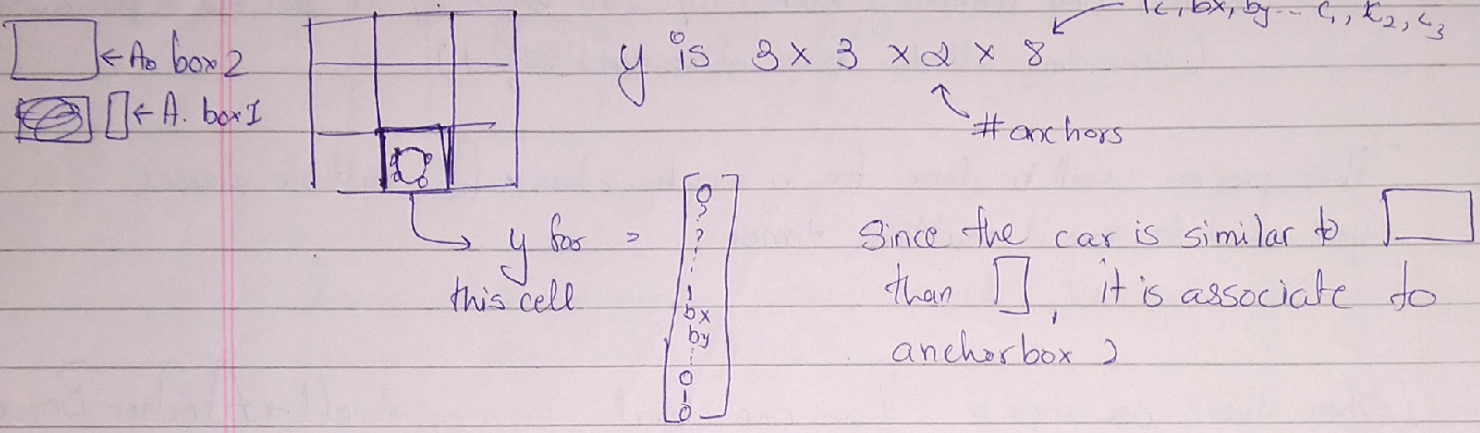
y: 3x3x8

With 2 anchor boxes
Each object in training image is assigned to grid cell that contains objects midpoint & anchor box for the grid cell with highest IoU

dim y: 3x3x16

This overlapping generally doesn't happen where m.p. of two objects is on the same place. As, ~~in real~~ we generally use 19x19 grid & not 3x3

Putting all back to YOLO



Region Proposals – R-CNN

They run a segmentation algorithm. This produces something like heat images (looks like it to me)
Here, the ones which have a different color or are highlighted ~~on~~ are only fed for object detection instead of the complete image.
Bounding boxes (square or rectangles) are draw on those highlighted objects.

R-CNN outputs one region (and bounding box) at a time and predicts the label. This makes it slower

Fast R-CNN Uses convolution implementation of sliding windows to classify all proposed regions. But, it was still slow

Faster R-CNN Use convolutional n/w to propose regions instead of segmentation algo. Runs a bit faster than fast R-CNN algo.
But this Faster R-CNN was a bit slower than YOLO Algorithm